Benutzerhandbuch
POSYS® 1800-B

PC104 1-4 Achsen Positioniersteuerkarte
für

DC/EC Servo-,
Mikroschritt- und Schrittmotoren

© POSYS Motion Control GmbH & Co.KG, 2014

Allgemeines

Über dieses Handbuch

Die POSYS®-Serie ist ein hoch entwickeltes Positioniersystem für DC/EC Servo-, Mikroschritt-, Schritt- und Piezomotoren als auch für Hydrauliksysteme. Die POSYS® 1800-B Serie ist eine PC104-kompatible Karte. Jede Karte ist in Konfigurationen für 2 oder 4 Achsen (1 oder 3 Achsen auf Anfrage) für Servo (bürstenbehaftete oder bürstenlose) und Schrittmotoren (Standard oder Mikroschritt) erhältlich (jede Achse frei-konfigurierbar oder in speziellen, abgespeckten Versionen mit eingeschränkten Möglichkeiten). Die Verwendung eines DSP zusammen mit einem ASIC sorgt zusammen mit dem 16-bit DAC-, 10-bit PWM oder Takt- & Richtungsausgangssignal für ausgezeichnete Positioniersteuerungsqualitäten. Zusammen mit den 16 frei-programmierbaren E/As gibt es außerdem noch dedizierte Eingänge (richtungsabhängige Endschalter mit einem digitalen Filter und High Speed Latch Eingänge für jede Achse). Des weiteren stehen 2 achsen-spezifische E/As und für jede Achse ein Ausgang für die Verstärkerfreigabe zur Verfügung. Diese E/As können auch für eigene Zwecke verwendet werden. Programmierbare Zeitnehmer könnten auch als Zähler eingesetzt werden. Ein digitaler PID- oder PI-Filter mit KP, KL, KD, Kvff und IL und der weite, programmierbare 16-Bit-Bereich sind zusätzliche Gründe für das überlegene Verhalten dieses Reglers. Ein weiteres Plus sind die hohe 8 MHz Inkremental- oder 160 MHz-Absolutenkodereingangsfrequenz und je nach Modell die 50 bis 75 µsec Servo-Loop-Updaterate pro Achse.

Die POSYS® 18xx-B Serie wurde entwickelt um höchste Flexibilität zu einem erschwinglichen Preis zu gewährleisten und um Antworten für komplexe Steuerungsprobleme zu bieten sowie die Peripheriekomponenten wie die digitalen und analogen Eingänge und die programmierbaren Ausgänge auf einfache Art zu handhaben. Die dedizierten Ausgänge für In-Position, Schleppabstandsfehlererkennung und Verstärkerfreigabe signalisieren den eigentlichen Status jeder Achse. Software-Kompatibilität wird durch alle Versionen garantiert (außer anders notiert; z.B. bei motorspezifischen Unterschieden).

Der POSYS®-Regler wurde für komplexe Positioniersteuerungsanwendungen wie u.a. Jogging, Punkt-zu-Punkt Positionierung, Vektorpositionierung, Electronic Gearing und für multiple Bewegungssequenzen (lineare und zirkulare (Option) Interpolation) entworfen. S-Kurven Profile eliminieren ruckartige positive und negative Beschleunigung; dies ist besonders nützlich, wenn schwere Trägheitslasten bewegt werden. Optionale Software-Pakete für die Ausführung zirkularer und linearer Interpolation für kontinuierliche Bewegungen erhöhen weiter die Einsatzmöglichkeiten dieser Positioniersteuerung. Beispielsoftware (z.B. PMD's Pro-Motion) mit einer Vielzahl unterschiedlicher textalischer und graphischer Ausgabemöglichkeiten als auch der Möglichkeit Hardcopies für Vergleichszwecke in der Performance zu erstellen, helfen dem Anwender die optimalen Filterparameter für jede mögliche Ladung und jeden Systemzustand zu ermitteln.

Inhaltsverzeichnis

Sicherheits & Gewährleistungshinweis 7
Gewährleistung 7
Sicherheitshinweis 7
Haftungsausschluß 7
Verwendete Symbole 8
Änderungen zu vorhergehenden Handbüchern 8
Blockschaltung 9
Blockschaltbild 9
Control Modules 10
Control Flow Überblick 10
Aktivieren und Deaktivieren von Control Modules 10
Reset Kommando 11
Zykluszeit setzen 12
Das Zeitregister 12
GetVersion Kommando 13
Bahnprofilerstellung 14
Bahnen, Profile und Parameter 14
Definition der Bahnparameter 14
Trapezfömiger Punkt-zu-Punkt Profil 15
S-Kurven Punkt-zu-Punkt Profil 17
Geschwindigkeitsmodus 19
Electronic Gear Profil 20
External Profile Mode 21
Relevante On-Board RAM-Funktionen 22
Profile Mode 22
Trace Mode 23
Verwendung von On-Board RAM für External Profile Mode 23
Was bedacht werden muß 24
Verfügbarer Speicher (RAM) zu klein für Programmausführung! Was tun? 24
Das SetStop Kommando 25
Aktivieren und Deaktivieren von Trajectory Generator Modules 25
Position Loop 26
Überblick 26
PID Loop 26
Integration Limit 28
Output Scaling 28
Derivative Sampling Time 28
Dual Encoder Support 29
Dual Encoder Support 29
Dual Encoder PID Loop Algorithmus 29
Konfiguration von Dual-Enkoder Unterstützung 30
Biquad Output Filter 31
Biquad Output Filter 31
Biquadkoeffizienten bestimmen 32
Bestimmung des Bi-Quad Skalierungsfaktors 32
Skalierung der Bi-Quad Koeffizienten 33
Output Limit 34
Output Limit 34
Motor Bias 35
Motor Bias 35
Deaktivieren und Aktivieren des Position Loop Moduls 36
Deaktivieren und Aktivieren des Position Loop Modules 36
Position Loop Werte lesen 37
Parameter Update and Breakpoints 38
Parameter Buffering 38
Updates 38

Breakpoints 39
Einen Breakpoint definieren 39
Breakpoint Trigger 40
Threshold-Triggered Beakpoints 41
Level-Triggered Breakpoints 41
Breakpoint Aktionen 41
Breakpoint Latenzen 42
Breakpoint Beispiele 42
Status Registers 45
Überblick 45
Event Status Register 45
Instruktionsfehler 46
Activity Status Register 46
Signal Status Register 47
Signal Sense Mask 48
Bewegungsüberwachung und verwandte Aufgaben 50
SetEventAction Prozess 50
Schleppabstandsfehler 51
Endschalter 51
Tracking Window 53
Motion Complete Indikator 54
In-Motion Indikator 55
Settle Window 55
Trace Capture 56
Trace Buffer Architektur 56
Der Trace Buffer 57
Der Trace Period 57
Trace Variablen 58
Trace Modes 59
Trace Start/Stop Bedingungen 59
Download von Trace Daten 60
Datenerfassung Starten 61
Host Interrupts 61
Hardware Control Signals 64
Hardware Control Signals 64
Das AxisOut-Signal 64
Der AxisIn-Eingang 64
Analoge Eingänge 64
The Synch Pin - Multiple Card Synchronization 65
Enkoderschnittstelle 66
Enkoderschnittstelle 66
Inkrementaler Enkodereingang 66
Aktuelle Positionsregister 66
Digitale Filterung 67
High-Speed Position Capture 68
High-Speed Position Capture 68
Absolutenkodereingang (SSI, BiSS, EnDat) 69
Absolutenkodereingang (SSI, BiSS, EnDat) 69
Multi-Turn Systeme 69
Motorausgabesignale 70
Motorausgabesignale 70
Motor Output Module deaktivieren 70
Aktivieren des Motor Output Modules 70
Motortyp 70
Motorausgabeformat 71
Sign/Magnitude PWM 72
50/50 PWM 72
16 Bit DAC Signalausgabe 73
Einstellung der PWM-Frequenz 73
Multi-Phasen Motoranbindung 74

Multi-Phasen Motorkommandointerpretation 75
Takt- & Richtungssignalausgabe 76
Mikroschrittmotorsignalausgabe 77
Motorausgangssignal Interpretation 77
Impuls-Breiten-Modulierte (PWM) Motorsignalausgabe 77
Motortreiber-Konfigurationen 78
Hostkommunikation 80
Hostkommunikation 80
Host I/O Commands 80
Paralleler Kommunikationsport (ISA-Bus) 80
Instruktionsfehler 81
Serielle Schnittstelle 82
Konfiguration 82
Kommandointerpretation 82
Instruktionsfehler 83
Checksummen 83
Übertragungsprotokolle 84
Point-to-Point Mode 84
Multi-drop/Idle-line Mode 84
Idle-line Mode 85
Address Bit Protocol 85
Controller Area Network (CAN) 86
Controller Area Network (CAN) 86
Überblick 86
Message Format 86
Konfiguration der CAN Schnittstelle 87
CAN Ereignis Benachrichtigung 88
EC Servomotorkontrolle 89
Überblick 89
Phasenanzahl 89
Phasenkontrollmodule 90
Phasenzählungen 90
Phaseninitialisierung 91
Hallsensor-basierte Phaseninitialisierung 91
Algorithmische Phaseninitialisierung 92
Direct-Set Phaseninitialisierung 93
Programmierung der Phaseninitialisierung 93
Hallsensor-basierte Initializierungssequenz 93
Algorithmische Initialisierungssequenz 94
Direct-Set Initialisierungssequenz 94
Index-Taktreferenzierung 94
Phasenfehlererkennung 95
Justierung der Phasenwinkel 95
Encoder Prescaler 96
Sinuskommutierung 98
Schrittmotorkontrolle 99
Überblick 99
Bahngeneratorkontrolleinheiten 99
Enkoderrückmeldung 100
Stall Detection 101
Takt & Richtungssignalgenerierung 101
Mikroschritt Motor Kontrolle 102
Mikroschrittsignalformen 103
Haltemoment Funktionen 104
On-Board Speicher: Definition und Zugriff 105
Speicherkonfiguration 105
PC104 ISA Bus (IO) 106
PC104 ISA Bus (Speicher) 106
MC Bus 106
Externe Speicherpuffer 106

On-Board Speicherkommandos 107
User I/O 108
Kommandoübersicht - ReadIO/WriteIO 108
Digital I/O-Kontrollregister (I/O space address +0) 108
Verstärker- & DAC Freigabekontrollregister (I/O space address +1) 108
Reset Überwachungskontrollregister (I/O space address +2) 109
Card ID-Kontrollregister (I/O space address +0xff) 109
Watchdog Kontrollregister (IO space address +4) 110

Sicherheits & Gewährleistungshinweis

Gewährleistung

POSYS Motion Control GmbH & Co.KG garantiert die Leistung seiner Produkte entsprechend den zur Verfügung stehenden Spezifikationen zum Zeitpunkt des Verkaufs für einen Zeitraum von 2 Jahren. Ebenso garantiert POSYS Motion Control GmbH & Co.KG für einen Zeitraum von 2 Jahren ab Verkaufsdatum die einwandfreie Materialbeschaffenheit unter der Voraussetzung, dass die Produkte innerhalb der gesetzten Spezifikationen betrieben werden und ein normaler täglicher Gebrauch unterstellt wird, d.h. Produkte die länger als 8 Stunden täglich Verwendung finden, können einer kürzeren Garantieperiode unterliegen. Für Details setzen Sie sich mit POSYS Motion Control GmbH & Co.KG in Verbindung.

POSYS Motion Control GmbH & Co.KG erlaubt sich Änderungen an seinen Produkten vornehmen zu können, oder Produkte oder Dienstleistungen einzustellen ohne vorhergehende Information und bittet Kunden und Interessenten die möglichst neuesten Informationen zu verifizieren ehe Bestellungen plaziert werden um sicher zu stellen, dass alle Informationen aktuell und komplett sind. Alle verkauften Produkte unterliegen den erweiterten Verkaufsbedingungen der ZVEI von 2002.

-o-

Sicherheitshinweis

Bestimmte Applikationen in denen elektronische Komponenten verwendet werden, können ein potenzielles Risiko für Leben und Material darstellen. Unsere Produkte sind nicht entwickelt worden bzw. wir authorisieren nicht die Verwendung unserer Produkte in Gerätschaften die der Lebenserhaltung dienen oder anderen kritischen Applikationen. Die Verwendung der Produkte der Firma POSYS Motion Control GmbH & Co.KG in solchen Gerätschaften bzw. in solch kritischen Applikationen geschieht ausser Garantie und auf volles Risiko des Kunden.

Um Risiken in Verbindung mit Applikationen des Kunden zu minimieren, müssen das Design und die operativen Sicherheitsmaßnahmen durch den Kunden sichergestellt sein um Gefahren abzuwehren.

-o-

Haftungsausschluß

POSYS Motion Control GmbH & Co.KG übernimmt keine Haftung für Anwendungshilfe oder Kundenproduktdesign. POSYS Motion Control GmbH & Co.KG übernimmt auch keine Haftung für Produktionsausfall, Schäden oder sonstige Kosten, ausser für Kosten die für die notwendige Reparatur oder Ersatz für Produkte der Firma POSYS Motion Control GmbH & Co.KG anfallen. Weitere Ersatzansprüche können nicht geltend gemacht werden.

-o-

Verwendete Symbole

Symbol	Bedeutung
⚠GEFAHR	Weist auf eine gefährliche Situation hin, die, wenn sie nicht vermieden wird, zum Tode oder zu schweren, irreversiblen Verletzungen führen wird.
⚠WARNUNG	Weist auf eine gefährliche Situation hin, die, wenn sie nicht vermieden wird, zum Tode oder zu schweren, irreversiblen Verletzungen führen kann.
⚠VORSICHT	Weist auf eine gefährliche Situation hin, die, wenn sie nicht vermieden wird, zu leichten Verletzungen führen kann.
HINWEIS	Weist auf eine Situation hin, die, wenn sie nicht vermieden wird, zu Beschädigung von Sachen führen kann.

Symbol	Bedeutung
	Dies ist kein Sicherheits-Symbol. Dieses Symbol weist auf wichtige Informationen hin.

Tabelle 1: Verwendete Symbole

-o-

Änderungen zu vorhergehenden Handbüchern

In diesem Bereich werden alle Änderungen bzgl. Hardware oder Software dokumentiert um schnell und zuverlässig wichtige Informationen zur Verfügung zu stellen, die evtl. den Betrieb der POSYS 1800-B sicherer gestalten, Fehler beheben, oder Erweiterungen beschreiben die hinzukommen.

Datum	Betreff	Beschreibung	Ab Handbuch #
21.05.2014	SSI Auflösung	Die Beschreibung der Auflösung für die SSI-Schnittstelle wurde von 11 Bit auf 12 Bit korrigiert	Enkoderschnittstelle

Tabelle 2: Anpassungen, Ergänzungen, Fehlerbereinigungen

-o-

Blockschaltung

Blockschaltbild

Das Blockschaltbild der Baugruppe POSYS 1800 B ist im Folgenden dargestellt. Der Motion Controller (MC) wird über die PC/104 ISA-Busschnittstelle 16 Bit breit gesteuert. Die Kommunikation zwischen ISA-Bus und MC wird über ein FPGA erledigt. Dem FPGA kommt dabei eine zentrale Bedeutung zu. Es bildet die Schnittstelle zwischen einer Vielzahl von Subsystemkomponenten und mehreren Bus Domains. Im Einzelnen sind folgende Funktionen realisiert:

PC/104 ISA-Bus <=> MC Host Port
PC/104 ISA-Bus <=> FPGA User Register
PC/104 ISA-Bus <=> DPRAM
PC/104 ISA-Bus => SPI DAC
MC CP <=> MC User Space
MC CP <=> DPRAM
MC CP => SPI DAC
MC CP <=> SSI Absolutdrehgeber

Die Stromversorgung erfolgt über den ISA-Anschluß des PC/104-Systems. Die Idle Leistungsaufnahme der betriebsbereiten Baugruppe mit 4-Achs Motion Controller beträgt weniger als 2,5W.

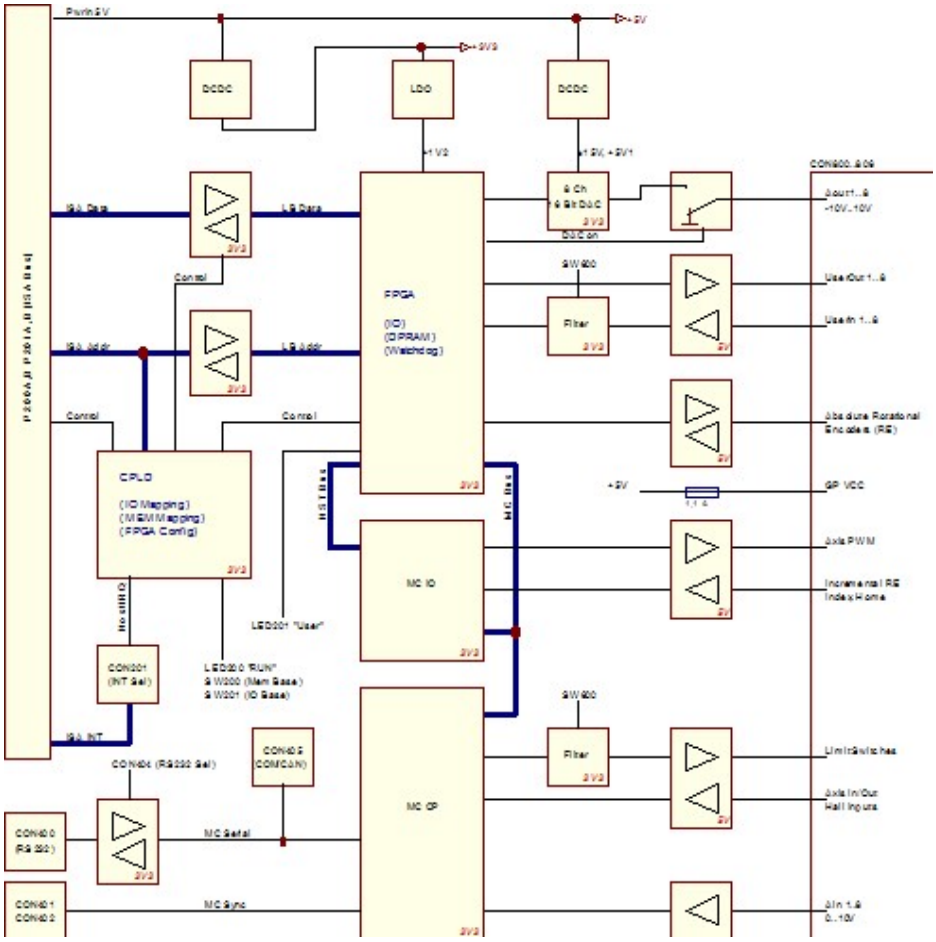

Abbildung 1: Blockschaltung POSYS 1800-B

-o-

Control Modules

Control Flow Überblick

Die folgende Grafik gibt einen Überblick über die Control Flow Funktionen der POSYS® Positioniersteuerkarten der 1800-B Serie. Es zeigt wie ein Motorkommandosignal generiert wird, beginnend mit der Profilauswahl und endend mit der eigentlichen Ausgabe des Motorausgabemoduls welches Verstärker-kompatible Sollwertvorgaben ausgibt. Abhängig vom Produkt und verwendetem Motortyp sind manche Module nicht verwendbar. z.B. ist im Schrittmotormodus die Auswahl des Position PID Loops nicht möglich.

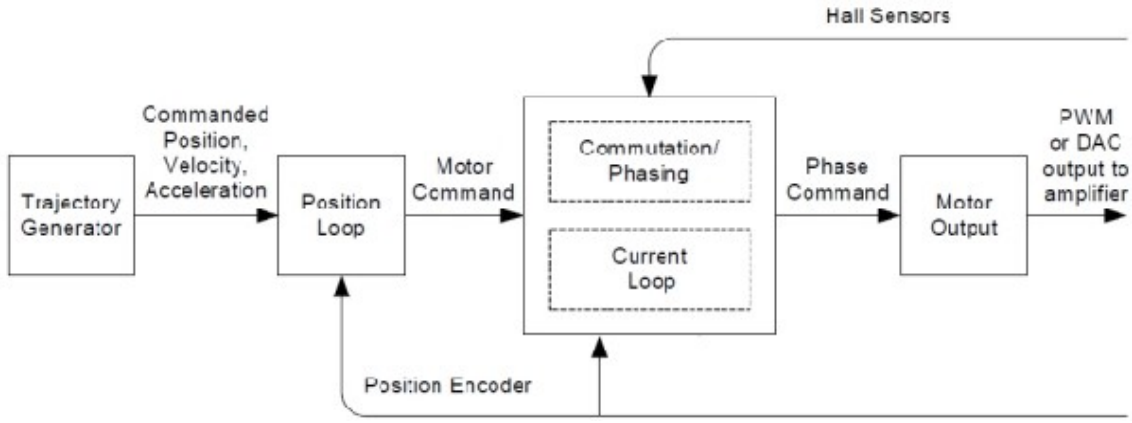

Abbildung 2: Control Flow Überblick

Jedes der Hauptblöcke im Control Flow Überblick ist assoziiert mit einem Modul. Die folgende Tabelle gibt einen kurzen Überblick der einzelnen Module und ihrer Bedeutung.

Modulename	Funktion
Trajectory Generator	Dieses Modul akzeptiert benutzer-definierte Eingaben und generiert daraus das Bahnprofil
Position Loop	Dieses Modul wird nur in Verbindung mit Servomotoren eingesetzt. Die Sollposition (momentane absolute gewünschte Position) und die Istposition (momentane tatsächliche Position) werden eingegeben und miteinander verglichen, woraus sich der Schleppabstandsfehler errechnet und dann durch den PID-Filter evtl. zusammen mit aktivierten Biquad-Filtern geschickt und generiert daraus eine Sollwertvorgabe für die Motorsignalausgabe.
Commutation/Phasing	Dieses Modul wird in Verbindung mit mehrphasigen Motoren verwendet (bürstenlose Servomotoren, Mikroschrittmotoren) und generiert für jede Phase die benötigte Sollwertvorgabe.
Motor Output	Dieses Modul übergibt das gewünschte Motorphasenkommando und erzeugt die entsprechenden Signale für das ausgewählte Ausgabeformat.

Tabelle 3: Modulname und Funktion

Jedes dieser Module ist in den nachfolgenden Kapiteln beschrieben. Darüberhinaus bieten die POSYS® 1800-B Positioniersteuerungen zusätzliche Funktionalitäten wie z.B. Breakpoints, Tracing, SPS-ähnliche Signalkontrolle usw. Diese Funktionen sind für alle Modelle gleichermaßen verfügbar und werden ebenso in späteren Kapiteln näher beschrieben.

-o-

Aktivieren und Deaktivieren von Control Modules

Beim Setup und im Betrieb kann es durchaus dem Wunsch entsprechen bestimmte Control Module zu aktivieren oder zu deaktivieren. Dies erfolgt mit dem Kommando SetOperatingMode. Um den mit diesem Kommando gesetzten Status zurückzulesen wird das Kommando GetOperatingMode verwendet. Generell gesagt: der Motion Controller überspringt sämtliche mit dem Modul assoziierten Kalkulationen und die Eingabe vom vorhergehenden Modul wird

direkt weitergeleitet zum nachfolgenden Modul ohne Änderungen.

Die folgende Tabelle erläutert welche Module deaktiviert und aktiviert werden können und beschreibt typische Umstände unter denen dies sinnvoll erscheint. Zusätzlich zu der Funktionalität einzelne Module zu aktivieren/deaktivieren können ganze Achsen völlig stillgelegt werden. Bitte beachten Sie dass das Kommutierungs- und Phasenmodul durch den Anwender nicht aktiviert oder deaktiviert werden kann. Falls ein bürstenloser Servomotor oder Mikroschrittmotor ausgewählt wurde, ist dieses Modul immer aktiviert.

Modul	Beschreibung	Typische Anwendung
Trajectory Generator	Falls deaktiviert verbleibt die Sollposition bei seinem gegenwärtigen Wert	Dieses Modul wird für gewöhnlich nicht manuell deaktiviert. Für die Bahnkontrolle welches einen sofortigen Stopp erfordert, würde das Kommando SetStopMode mit dem Argument *Abrupt Stop* stattdessen verwendet.
Position Loop	Wird nur in Verbindung mit Servomotoren verwendet. Falls deaktiviert, gibt das Modul Ausgaben aus zwei Quellen abhängig davon ob das Modul aktiv oder deaktiv ist. *Trajectory Module enabled*: Falls aktiviert, wird der Position Loop übersprungen und die Ausgabe des Profile Generators direkt in das nachfolgende Modul eingegeben. *Trajectory Module disabled*: Falls deaktiviert, kommt die Ausgabe vom Motor Command Register, welches mit dem Kommando SetMotorCommand manuell eingegeben werden kann. Im Kapitel "Deaktivieren und Aktivieren des Position Loop Modules" werden hierzu weitere Informationen bereitgestellt.	Das Deaktivieren dieses Modules während der Trajectory Generator aktiviert ist, ist sinnvoll falls spannungs- oder stromporpotional agierende Rückmeldeeinheiten (z.B. Galvanometer) verwendet werden. Es kann auch sinnvoll sein für die Verstärkerkalibrierung falls benötigt. Zusammen mit deaktiviertem Trajectory Generator und deaktiviertem Position Loop Module ist dies sinnvoll für Verstärker- und Motorkalibrierung.
Motor Output	Die Deaktivierung dieses Moduls hat zur Folge, dass alle Werte für die Motor-spezifischen Ausgaben auf 0 (Null) gesetzt werden. Der aktuelle Zustand der assoziierten Motorausgabesignale wird von der gewählten Ausgabemethode abhängen (PWM 50/0, PWM s/m, DAC)	Die Deaktivierung der Motorsignalausgabe kann in Verbindung mit sicherheits-relevanten Bedingungen oder für die Verstärkerkalibrierung sinnvoll sein.

Tabelle 4: Control Modules

Zusätzlich zur manuellen Deaktivierung verschiedener Kontrollmodule gibt es eine Reihe Bedingungen unter denen die automatische Deaktivierung sinnvoll sein kann, wie z.B. Ereignis-bedingte oder durch Breakpoints bestimmte Bedingungen.

GetOperatingMode liefert den programmierten Status zurück der mit dem Kommando SetOperatingMode den gewünschten Operationsmodus für normale Bedingungen setzt. Jedoch kann dies vom aktuellen Operationsmodus aufgrund oben angeführter Gründe abweichen. Um den aktuellen Status abzufragen wird das Kommando GetActiveOperatingMode verwendet.

-o-

Reset Kommando

Zusätzlich zur Aktivierung und Deaktivierung des Control Modules, ist es möglich den Motion Controller völlig zu resetten und zwar mit Hilfe des Reset-Kommandos. Dieses Kommando setzt alle Register auf ihre Standardwerte zurück und reinitialisiert alle Motion Control-Funktionen. Siehe POSYS 1800-B Programmerierhandbuch für Details welche Register auf welche Standardwerte zurückgesetzt werden.

Ein Reset-Kommando hat die gleiche Wirkung wie das Betätigen des Hardwareschalters zur Aktivierung eines Hardware-Resets. Neben dem manuellen Resetten oder dem signalbasierten Resetten, erfolgt auch ein Reset während des Einschaltens. Aufgrund der großen Anzahl von erforderlichen Operationen, dauert ein Reset-Vorgang wesentlich länger als das normale Abarbeiten von Standard-Kommandos.

Beachten Sie, dass im normalen Betrieb die Ausführung eines Resets nicht erforderlich ist. Im Allgemeinen wird es

nur während der Entwicklung oder zur Fehlersuche benötigt, um das System wieder in einen bekannten Ausgangszustand zu bringen.

Das Ausführen eines Reset-Kommandos führt dazu die Motorsollwertvorgabe für alle Achsen sofort auf Null (0) zu setzen, und alle Motion Controller Aktivitäten werden eingestellt. Der Controller ist wieder in den Zustand zurückversetzt in dem er sich befindet wie nach einem Neustart des Systems. Es liegt in der Verantwortung des Anwenders zu ermitteln, ob die Ausführung eines Reset-Kommandos sicher ist für einen bestimmten Betriebszustand in dem sich das Komplettsystem gerade befindet.

-o-

Zykluszeit setzen

Die POSYS® kalkuliert jede Bahn und Servoinformation in einem festen Zeitabstand. Dieser Abstand wird die Zykluszeit (cycle time) genannt. Für jede aktivierte Achse der POSYS® gibt es eine minimale Zykluszeit, damit die POSYS® alle anfallenden Kalkulationen erledigen kann. Die minimale Zykluszeit für eine POSYS® 18xx-B ist 102,4 µsec, abhängig von der Anzahl Achsen und Motorvarianten die möglicherweise angeschlossen sind. Mit dieser Einstellung ist die bestmögliche Frequenz 9.76 kHz. Diese Frequenz kann in 64/68,3/76,8/102,4 µsec Schritten erhöht werden. Die folgende Tabelle zeigt die Unterschiede:

# aktivierte Achsen	Minimale Zykluszeit	Zykluszeit m/ aktivierter Tracingfunktion	Zykluszeit pro Achse	Minimale Zyklusfrequenz
1	102.4 µs	102.4 µs	102.4 µs	9.76 kHz
2	153.6 µs	153.6 µs	76.8 µs	6.51 kHz
3	204.8 µs	204.8 µs	68.3 µs	4.88 kHz
4	256 µs	256 µs	64 µs	3.91 kHz

Tabelle 5: Zykluszeit (Servo-Loop-Updaterate) mit aktivierten Achsen

Die Zykluszeit bestimmt die Servo-Loop-Updaterate anhand der die Parameter pro aktivierte Achse neu kalkuliert und angeglichen werden. Jedoch bestimmt es nicht die Kommutierungsrate für bürstenlose Servomotoren.

Jede aktivierte Achse erhält sein „Zeitfenster“ egal ob es in Bewegung ist oder nicht, egal ob mit dem Kommando SetMotorMode im EIN oder AUS Zustand ist. Falls die Zykluszeit kritisch ist, ist es möglich die Zykluszeit zu verkürzen in dem unbenutzte Achsen deaktiviert werden (SetAxisMode) und in dem die Servo-Loop-Updaterate mit dem Kommando SetSampleTime neu gesetzt wird.

z.B., bei Verwendung der POSYS® ist es möglich mit 4 Achsen zu operieren, aber falls in einer spezifischen Anwendung nur 3 Achsen benötigt werden, kann die vierte Achse mit dem Kommando SetAxisMode oder dem neuen Kommando SetOperatingMode deaktiviert und eine neue Zykluszeit von 204,8 µs gesetzt werden. Dies verbessert die Frequenz von 3,9 kHz auf 4,88 kHz.

SetSampleTime kann auch verwendet werden um die Zykluszeit zu verlängern, falls dies notwendig sein sollte.

Es liegt in der Verantwortung des Anwenders dafür Sorge zu tragen, dass die Zykluszeit gleich oder größer als die spezifizierten Minimalwerte, wie in der Tabelle voraus beschrieben, sind. Werte die kleiner als die kleinstmöglichen Einstellungen sind, werden automatisch auf die Minimalwerte zurückgesetzt.

-o-

Das Zeitregister

Die POSYS Controller verfügen über einen 32-Bit-Register, welches die aktuelle Motion Controller Zeit hält, in Form von der Anzahl Zyklen die ausgeführt wurden, seit dem Einschalten bzw. einem Reset aufgelaufen sind. Dieser ständig wechselnder Wert kann mit dem Kommando GetTime gelesen werden.

Das Register hat zwei primäre Zwecke. Es kann als Vergleichswert für zeit-basierte Haltepunkte verwendet werden (siehe Abschnitt "Breakpoints", für Details). Darüber hinaus können sie ein nützliches Instrument zur Verfolgung der tatsächlich benötigten Zeit für bestimmte Aufgaben eingesetzt werden.

Das Zeit-Register erhöht sich um einen Wert von 1 für jeden Zyklus, den der Motion-Controller bis zu ihrem

größtmöglichen Wert von FFFF FFFFh oder 4294967295 dec, an welcher es auf Null (0) zurück springt. Der Punkt, an dem der Zeitrücksprung auftreten wird, hängt von der eingestellten Zykluszeit für den Motion-Controller ab. Zum Beispiel für einen 4-Achsen POSYS 1824-B mit eingestellter Standard-Zykluszeit von 256 µSec, erfolgt der Rücksprung nach 256 * 4294967295 uSec = ~ 12,7 Tagen. Alle Motion Controller Operationen werden in der Regel weiter arbeiten, aber wenn ein Zeit-definierter Breakpoint gesetzt wurde, sollte darauf geachtet werden dass der Vergleichswert korrekt errechnet wurde inkl. möglicher Übersprünge.

-o-

GetVersion Kommando

Dieses Kommando und die zurückgelieferte Nummer liefern keine Information darüber ob eine PCI-, ISA- oder PC/104- oder andersformatige (z.B. Kundenspezifische) Karte eingesetzt wird zurück, sondern nur die interne Funktionsinformation des eingesetzten Motion Control Chipsatzes. Für Informationen bzgl. der eingesetzten Karte verwenden Sie das Kommando ReadPLD.

-o-

Bahnprofilerstellung

Bahnen, Profile und Parameter

Der Bahngenerator kalkuliert die aktuelle Position, Geschwindigkeit und Beschleunigung zu einer gegebenen Zeit. Diese Werte werden die Sollwerte genannt. Während einer Bewegungssequenz werden sich einige oder alle dieser Parameter kontinuierlich ändern. Sobald die Bewegungssequenz vollständig abgearbeitet ist, werden diese Parameter aktuell bleiben, bis eine neue Bewegung begonnen wird und mit ihr eventuell auch neue Parameter übergeben werden.

Um die aktuellen Sollprofilwerte abzufragen, werden die Befehle GetCommandedPosition, GetCommandedVelocity, und GetCommandedAcceleration benutzt.

Durch das ganze Handbuch hindurch werden verschiedene mnemotechnische Kommandos gezeigt um die Verwendung der POSYS® zu erklären oder spezifische Beispiele zu liefern. Der Programmierabschnitt liefert mehr Informationen über Befehle, Nomenklatur und Syntax.

Das spezifische Profil, dass die POSYS® einrichtet, hängt von einigen Faktoren einschließlich des aktuellen Profilmodus, der aktuellen Profilparameter, und anderer Systemzustände ab, wie z.B. ob ein Bewegungsstopp ausgelöst worden ist. Fünf Bahnprofilmodi werden unterstützt:

- S-Kurve Punkt-zu-Punkt.
- Trapezförmig (linear) Punkt-zu-Punkt.
- Geschwindigkeitsregelung
- Electronic Gearing (Folge-/Leitbetrieb).
- External Profile Mode

Die Verwendung dieser Profilmodi wird ausführlich in nachfolgenden Abschnitten erklärt. Der verwendete Befehl, den Profilmodus zu selektieren, ist SetProfileMode. Der Befehl GetProfileMode liefert den programmierten Profilmodus zurück.

Der Profilmodus kann unabhängig für jede Achse programmiert werden. Zum Beispiel Achse #1 kann im trapezförmigem Modus sein während Achse #2 für den S-Kurvenmodus programmiert wurde.

Mit einer Ausnahme kann die POSYS® von einem Profil in das andere schalten, während eine Achse in Bewegung ist. Die Ausnahme: um in den S-Kurven Punkt-zu-Punkt Modus zu schalten muß die Achse stillstehen. Es ist auch nicht möglich, während eine Bewegung noch abläuft, vom S-Kurvenmodus in einen anderen Modus zu schalten.

-o-

Definition der Bahnparameter

Die POSYS® sendet und empfängt Bahnprofildaten und verwendet dabei eine Parameterübergabe nach Fixpunktdarstellung. In anderen Worten es werden eine feste Anzahl Bits verwendet um den Integerwert einer Realzahl darzustellen.

Folgende 3 Formate verwendet die POSYS®:

Format	Wortgröße	Bereich	Beschreibung
32.0	32 bits	-2,147,483,648 to +2,147,483,647	Unity Scaling. Dieses Format verwendet nur Integerzahlen.
16.16	32 bits	-32,768 to +32,767 +65,535/65,536	Verwendet $1/2^{16}$ Skalierung. Die POSYS® erwartet eine 32-Bit Nummer welches mit einem Faktor von 65.535 skaliert wurde. z.B. um eine Geschwindigkeit von 2,75 zu definieren, muss 2,75 mit 65.535 multipliziert werden und das Resultat als 32-Bit Integer an die POSYS® geschickt werden (180.224 dez. oder 2c000h).
0.32	32 bits	0 to +2,147,483,647/4,294,967,296	Verwendet $1/2^{32}$ Skalierung. Die POSYS® erwartet eine 32-Bit Nummer welches mit einem Faktor von

Format	Wortgröße	Bereich	Beschreibung
			4.294.967.296 (2^{32}) skaliert wurde. z.B. um einen Wert von 0,0075 zu übergeben, muss 0,0075 mit 4.294.967.296 (2^{32}) multipliziert werden und das Resultat als 32-Bit Integer an die POSYS® geschickt werden (32.212.256 dez. oder 1eb8520h).

Tabelle 6: Bahnparameterdefinition

Die Umwandlung der Daten in die benötigten Formate wird bereits durch die mitgelieferte DLL erledigt.

-o-

Trapezfömiger Punkt-zu-Punkt Profil

Die folgende Tabelle umfasst die spezifischen Profilparameter für den trapezförmigen Punkt-zu-Punkt Modus:

Profilparameter	Format	Wortgröße	Bereich
Position	32.0	32 bits	-2,147,483,648 to 2,147,483,647
Starting Velocity	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle
Velocity	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle
Acceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2
Deceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2

Tabelle 7: Trapez-förmige Punkt-zu-Punkt Profilparameter

Die Kommandos SetPosition, SetVelocity, SetAcceleration, und SetDeceleration laden diese Werte. Die Befehle GetPosition, GetVelocity, GetAcceleration, und GetDeceleration liefern die programmierten Werte zurück.

Für dieses Profil errechnet der Host eine anfängliche positive und negative Beschleunigung, eine Geschwindigkeit, und eine Endposition. Das Profil bekommt seinen Namen aufgrund der Kurvenform (Abbildung 3): die Achse beschleunigt linear (anhand des programmierten Beschleunigungswertes), bis es die programmierte Geschwindigkeit erreicht. Die Geschwindigkeit bleibt erhalten, bremst dann linear ab (den negativen Beschleunigungswert benutzend, falls eine programmiert wurde), bis es an der angegebenen Position stehen bleibt. Falls die negative Beschleunigung einsetzen muss ehe die Achse die programmierte Geschwindigkeit erreicht, wird das Profil kein konstantes Geschwindigkeitsteil ausweisen, und das Trapez wird ein Dreieck (siehe Abbildung 5).

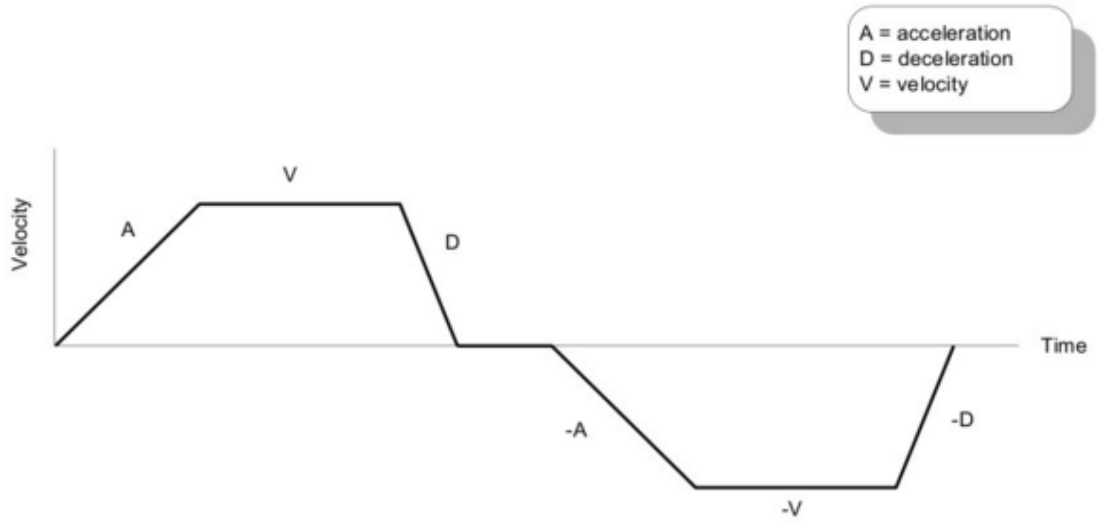

Abbildung 3: Einfaches trapez-förmiges Punkt-zu-Punkt Profil

Abbildung 4 illustriert ein trapez-förmiges Profil mit vorbestimmte Startgeschwindigkeit Null (0). Bei Schrittmotoren mit Einstellung „Vollschritt“ ist es gelegentlich wünschenswert eine Startgeschwindigkeit zu definieren von der aus

der Motor unverzüglich seine Bewegung startet. Dies ist nützlich um den resonanz-gefährdeten Bereich von 70-120 Schritte pro Sekunde zu überspringen/vermeiden. In der Phase der negativen Beschleunigung des Profils wird anstelle einer kontinuierlichen Abbremsrampe bis Null (0), der Motor unverzüglich bei Erreichen der Startgeschwindigkeit die Geschwindigkeit auf Null (0) reduzieren, ohne einer weiteren Abbremsrampe. Abbildung 4 zeigt ein solch typisches Profil mit eingestellte Startgeschwindigkeit.

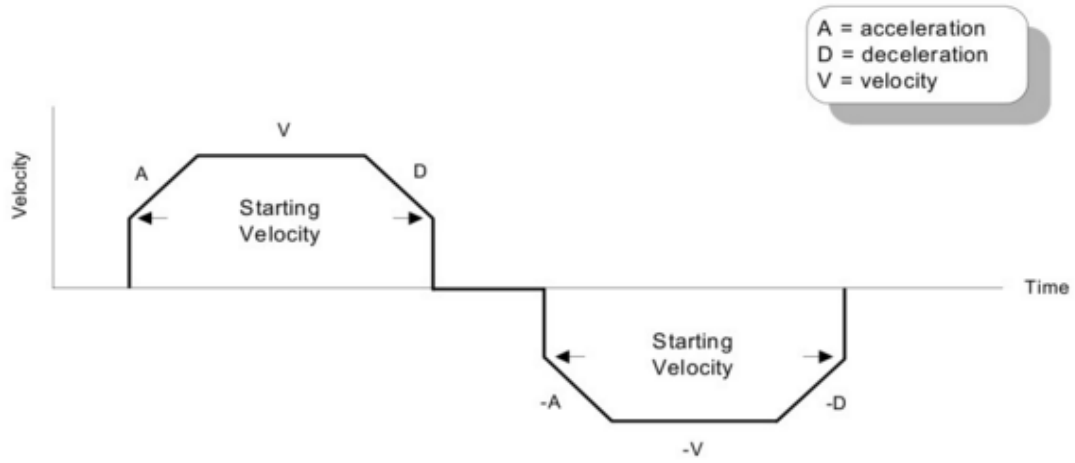

Abbildung 4: Trapez-förmiges Profil mit eingestellte Startgeschwindigkeit <> Null (0)

Anmerkung: Eine Startgeschwindigkeit kann nur im Trapezmodus (trapezoidal profile mode) und Geschwindigkeitsmodus (velocity contouring profile mode) eingestellt werden. Es ist nicht verfügbar für die Modi S-Kurve oder Electronic Gearing Profile Mode.

Falls die negative Beschleunigung einsetzen muss ehe die Achse die programmierte Geschwindigkeit erreicht, wird das Profil kein konstantes Geschwindigkeitsteil ausweisen, und das Trapez wird zum Dreieck (Abbildung 5).

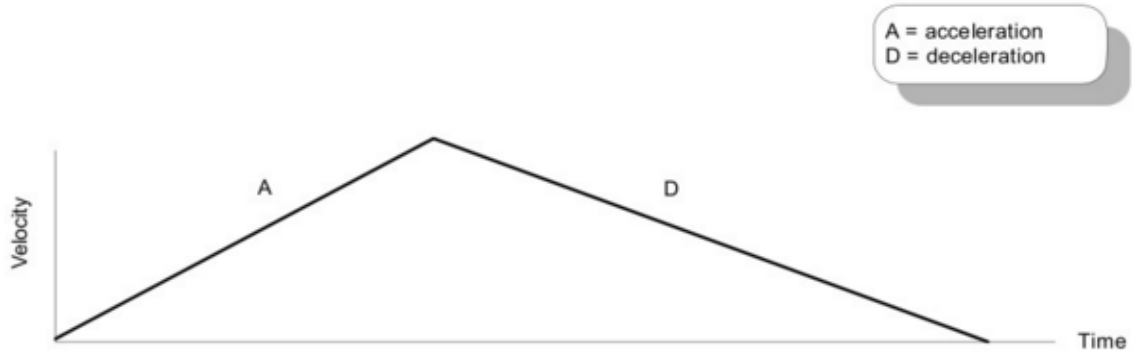

Abbildung 5: Einfaches trapez-förmiges Punkt-zu-Punkt Profil ohne konstanten Geschwindigkeitsanteil

Die Rampen können je nachdem ob und wenn ja auch die negative Beschleunigung programmiert wurde, entweder symmetrisch oder asymetrisch sein.

Der Beschleunigungsparameter wird immer am Anfang der Bewegungssequenz benutzt. Danach wird der Wert für die positive Beschleunigung in jene Richtung verwendet, und der Wert für die negative Beschleunigung wird für die entgegengesetzte Richtung eingesetzt. Falls keine Bewegungsparameter während der Bewegungssequenz verändert werden, dann wird der positive Beschleunigungswert verwendet bis die maximale Geschwindigkeit erreicht wurde und der negative Beschleunigungswert für die Abbremsrampe bis Null (0) erreicht wurde.

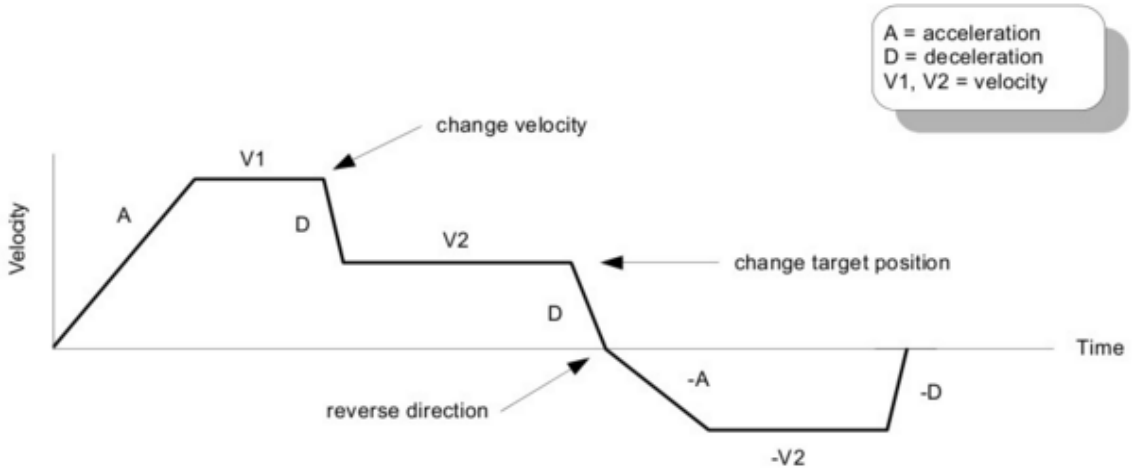

Abbildung 6: Komplexes trapez-förmiges Punkt-zu-Punkt Profil mit sich verändernden Parametern

Es ist möglich, einen der Profilparameter zu verändern, während die Achse sich in diesem Profilmodus befindet. Der Profilgenerator wird immer versuchen, die Bewegung innerhalb der durch die Parameter vorgegebenen Bedingungen auszuführen. Wird während der Bewegung die Endposition in solch einer Weise verändert, dass der Schleppabstand überschritten wird, wird die POSYS® mit Rampe bis zum Stopp abbremsen und dann in entgegengesetzte Richtung beschleunigen, um sich zu der neuen angegebenen Position zu bewegen. Dies wird in Abbildung 6 dargestellt.

Falls keine negative Beschleunigung programmiert wurde, oder = 0 ist, dann wird der Wert verwendet der für die positive Beschleunigung programmiert wurde.

-o-

S-Kurven Punkt-zu-Punkt Profil

Die folgende Tabelle fasst die host-spezifischen Profilparameter zusammen für den S-Kurven Punkt-zu-Punkt Modus:

Profilparameter	Format	Wortgröße	Bereich
Position	32.0	32 bits	-2,147,483,648 to 2,147,483,647 counts
Jerk	0.32	32 bits	0 to 2,147,483,647/4,294,967,296 counts/cycle^3
Velocity	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle
Acceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2
Deceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2

Tabelle 8: S-Kurve Punkt-zu-Punkt Profilparameter

Die Kommandos SetPosition, SetVelocity, SetAcceleration, SetDeceleration, und SetJerk laden die jeweiligen Werte. Die Befehle GetPosition, GetVelocity, GetAcceleration, GetDeceleration, und GetJerk liefern die programmierten Werte zurück. Das S-Kurven Punkt-zu-Punkt Profil fügt eine Begrenzung zum Verhältnis der Änderung der Beschleunigung hinzu, verglichen mit der Basis des Trapezprofils. Ein neuer Parameter (Jerk) spezifiziert die maximale Änderung der Beschleunigung innerhalb eines Updatezyklus.

Im S-Kurven Profilmodus, muss der gleiche Wert sowohl für Beschleunigungs- als auch für die Abbremsrampe benutzt werden. Asymmetrische Profile sind nicht erlaubt.

Das S-Kurven Punkt-zu-Punkt Profil fügt einen weiteren Parameter hinzu der den Faktor der Änderung der Beschleunigung definiert. Der Parameter (Jerk) spezifiziert die maximale Änderung der Beschleunigung pro Servoloop-Updaterate.

In diesem Profilmodus wächst die Beschleunigung allmählich von 0 zu dem programmierten Beschleunigungswert an, dann vermindert sich die Beschleunigung im gleichen Verhältnis, bis es 0 mit der programmierten Geschwindigkeit erreicht. Die gleiche Sequenz wird ausgeführt um die Endposition zu erreichen.

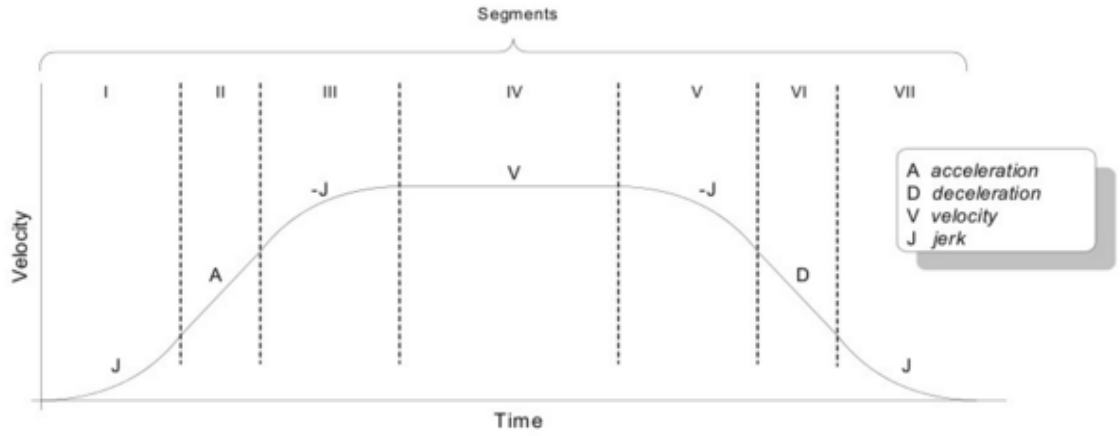

Abbildung 7: Typisches S-Kurven Punkt-zu-Punkt Profil

Abbildung 7 zeigt ein typisches S-Kurven Profil. In Segment I erhöht sich der Beschleunigungswert um den mit dem Befehl Jerk gesetzten Wert bis die maximale Beschleunigung erreicht wurde. Die Achse fährt fort, sich linear (jerk = 0) durch Segment II zu beschleunigen. Das Profil wendet dann im Segment III den negativen Wert des Jerks an um die Beschleunigung zu reduzieren. Im Segment IV verfährt die Achse jetzt mit maximaler (programmierter) Geschwindigkeit (V). Das Profil wird dann in einer dem Beschleunigungswert ähnlichen Weise abbremsen, indem in umgekehrter Richtung der negative Jerk verwendet wird, um zuerst die maximale Verlangsamung zu erreichen (A), und dann die Achse zu einem Halt an der Endposition zu bringen.

Eine S-Kurve mag vielleicht nicht alle Elemente der Abbildung 7 gezeigten Segmente enthalten. Zum Beispiel falls nicht die maximale Beschleunigung vor dem "Halbweg" in Richtung Endgeschwindigkeit oder Endposition erreicht werden kann, würde das Profil nicht die Segmente II und VI enthalten. Solch ein Profil wird in Abbildung 8 gezeigt.

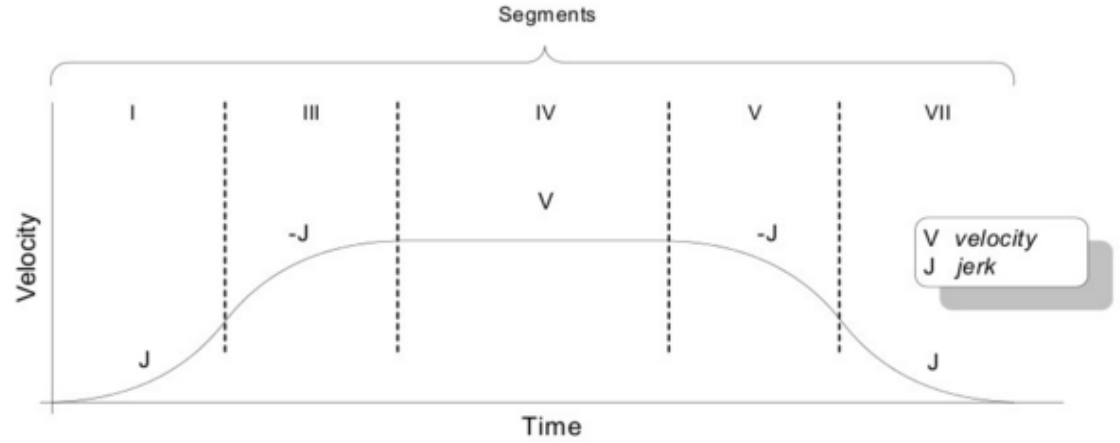

Abbildung 8: S-Kurve welches nicht die maximale Beschleunigung erreicht

Ähnlicherweise, falls eine Position derart angegeben wird, dass die Endgeschwindigkeit nicht erreicht werden kann, wird es kein Segment IV, wie in Abbildung 9 gezeigt, geben. (Es ist auch möglich, dass es auch kein Segment II oder Segment VI gibt, je nachdem wo das Profil "gekürzt" wird.)

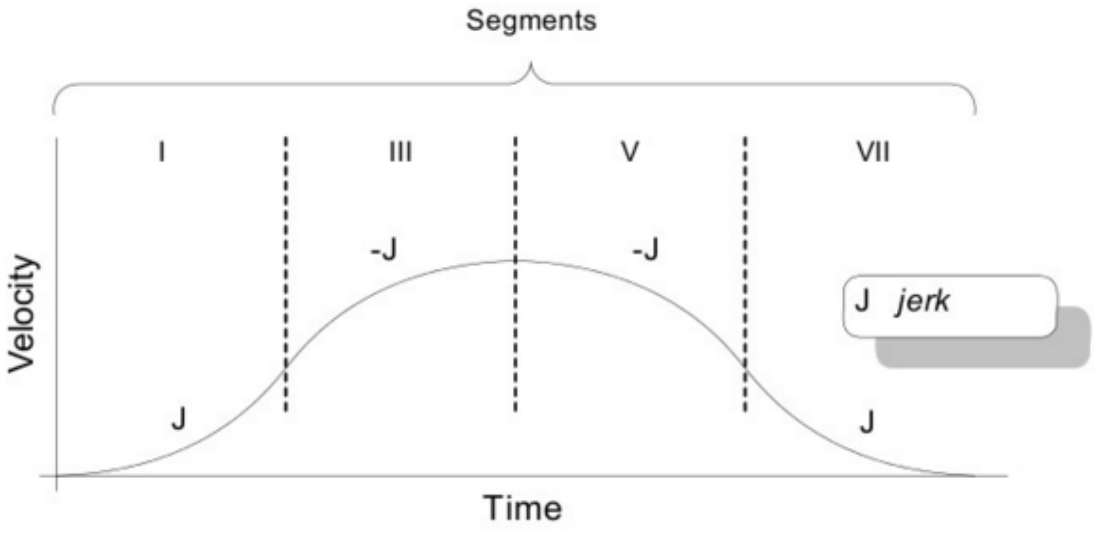

Abbildung 9: S-Kurve ohne maximalem Geschwindigkeitssegment

Im Gegensatz zum trapezförmigen Profilmodus, erlaubt der S-Kurven Profilmodus keine Änderungen an einem der Profilparameter während die Achse in Bewegung ist. Ebenfalls darf die Achse nicht in den S-Kurven Modus geschaltet werden, während die Achse in Bewegung ist.

Es ist allerdings erlaubt vom S-Kurven Modus zu einem anderen Profilmodus während der Bewegung zu wechseln. Falls der S-Kurven Modus für die Art der auszuführenden Bewegung notwendig ist und die Notwendigkeit besteht PID-Filter-, Geschwindigkeit- und/oder Beschleunigungswerte während der Bewegung zu ändern, dann ist es empfehlenswert in den Trapezmodus zu wechseln.

-o-

Geschwindigkeitsmodus

Die folgende Tabelle fasst die host-spezifischen Profilparameter zusammen für den Geschwindigkeitsmodus:

Profilparameter	Format	Wortgröße	Bereich
Start Velocity	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle
Velocity	16.16	32 bits	-32,768 to 32,767 + 65,535/65,536 counts/cycle
Acceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle²
Deceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle²

Tabelle 9: Profilparameter für den Geschwindigkeitsmodus

Die Kommandos SetVelocity, SetAcceleration, und SetDeceleration laden die jeweiligen Werte. Die Befehle GetVelocity, GetAcceleration, und GetDeceleration, liefern die programmierten Werte zurück.

Im Gegensatz zu den trapezförmigen und S-Kurven Profilmodi, welche die Endposition bestimmt welche Richtung positiv oder negativ wird, bestimmt das Vorzeichen im Geschwindigkeitsmodus die positive oder negative Richtung. Deswegen kann der Geschwindigkeitswert, der zur POSYS® gesandt wird, positive Werte (für eine positive Richtungsbewegung) oder negative Werte haben (für eine entgegengesetzte Richtungsbewegung).

In diesem Profil wird keine Endposition angegeben. Die Bewegung wird gänzlich kontrolliert, indem man die positive/negative Geschwindigkeit und Beschleunigung verändert, während das Profil ausgeführt wird.

Im Geschwindigkeitsmodus ist die Achsenbewegung nicht an eine Endposition gebunden. Es liegt in der Verantwortung des Anwenders, Geschwindigkeits- und Beschleunigungswerte zu verwenden die einen sicheren Bewegungsablauf garantieren.

Die Bahn wird ausgeführt, indem man kontinuierlich die Achse mit dem angegebenen Wert beschleunigt, bis die Geschwindigkeit erreicht wird. Die Achse fängt an abzubremsen, wenn eine neue Geschwindigkeit angegeben wird der einen kleineren Wert hat als die aktuelle Geschwindigkeit, oder ein anderes Vorzeichen hat als die aktuelle

Richtung vorgibt.

Ein einfaches Geschwindigkeitsprofil sieht aus wie ein einfaches trapezförmiges Punkt-zu-Punkt Profil, wie in Abbildung 3 dargestellt.

Abbildung 10 zeigt ein komplizierteres Profil, in dem beides, die Geschwindigkeit als auch die Bewegungsrichtung zweimal wechseln.

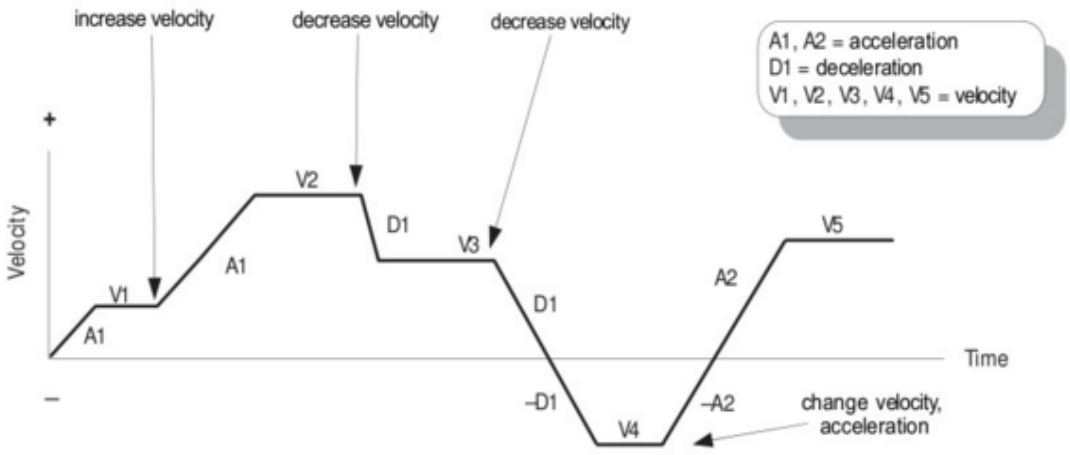

Abbildung 10: Geschwindigkeitsprofil

-o-

Electronic Gear Profil

Die folgende Tabelle fasst die host-spezifischen Profilparameter zusammen für den Electronic-Gear Modus:

Profilparameter	Format	Wortgröße	Bereich
Gear Ratio	16.16	32 bits	-32,768 to 32,767 + 65,535/65,536 counts/cycle
Master Axis #	-	2 bits	0-3*
Master Source	-	1 bit	2 Werte; aktuelle oder Sollposition (siehe unten für Details)

Tabelle 10: Profilparameter für Electronic Gearing

*für 2-Achsen Versionen 0-1. 1-Achsen Karten unterstützen kein Electronic-Gear Modus.

Die Kommandos SetGearRatio, und SetGearMaster laden die jeweiligen Werte. Die Befehle GetGearRatio und GetGearMaster liefern die programmierten Werte zurück.

In diesem Profil gibt der Host drei Parameter an. Der erste Parameter ist die „Master"-Achse #, die die Achse ist, die die Quellinformationen liefert um die „Slave"-Achse zu führen, die die Achse in Getriebemodus ist. Der Zweite ist die Gearsource die den Wert liefert nach der die Slave-Achse folgen soll. Es können entweder die Ist- oder Soll-Position sein. Der Dritte ist das Übersetzungsverhältnis, das angibt in welcher Richtung und mit welchem Verhältnis die Slave-Achse der Master-Achse folgen soll.

Die Folgeachse wird in der Regel eine andere Achse als die Leitachse sein. Die eine erlaubte Ausnahme ist bei Verwendung von Schrittmotoren. In diesem Fall kann die Leitachse auch als Folgeachse definiert werden, so lange als Quelle für das Folgeverhältnis auf Enkoder gesetzt wurde.

Abbildung 11 zeigt die Anordnung von Enkoder und Verstärker in einer typischen Electronic-Gearing Einstellung.

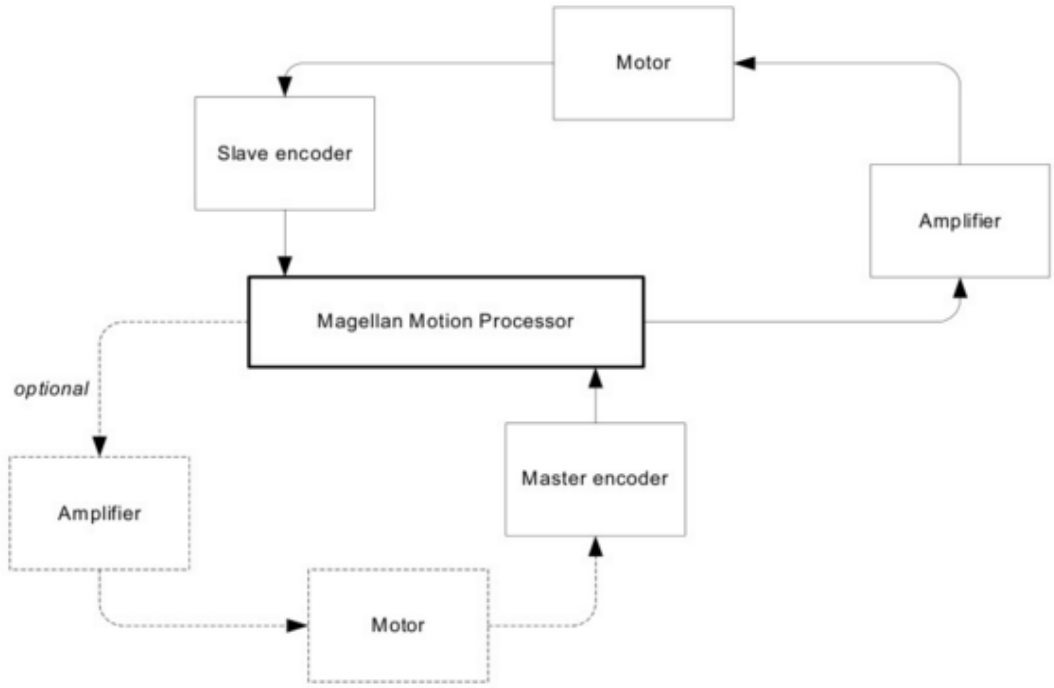

Abbildung 11: Electronic Gear

Ein positives Übersetzungsverhältnis bedeutet, dass wenn die aktuelle oder Soll-Position der Master-Achse sich in positiver Richtung erhöht, sich auch die Slave-Achse in positiver Richtung bewegt. Ein negatives Verhältnis hat den umgekehrten Effekt. Während die Position der Master-Achse sich erhöht verringert sich die Position der Slave-Achse.

Z.B., lassen Sie uns annehmen die Slave-Achse ist die Achse #0 (Achsen werden von 0 bis 3 bei einer 4-Achsen Karte gezählt) und die Master-Achse ist die Achse #3. Lassen Sie uns auch annehmen die Gearsource wird auf aktuelle Position mit einem Verhältnis von -½ gesetzt.

Der Electronic-Gearmodus erfordert, dass 2 Achsen definiert werden. Deswegen unterstützen die 1-Achsen Karten den Electronic-Gearmodus nicht. Falls die Gearsource der Master-Achse auf "Ist-Position" eingestellt wurde, ist es nicht notwendig einen Motor an dieser Achse zu betreiben. In der Hauptsache wird es als Enkodereingangszähler verwendet, z.B. im Open-Loop Modus oder als manuelle Einstellmöglichkeit für andere Achsen. Es ist allerdings auch möglich einen Motor zu verwenden, dabei muss dieser Achse ein anderer Profil zugewiesen werden. Der Effekt dieser Einstellung ist, dass beide Achsen, Master als auch Slave, mit dem selben Profil betrieben werden können, auch wenn der Slave mit einem anderen Übersetzungsverhältnis zum Master betrieben wird oder in eine andere Richtung verfährt. Für die Master-Achse ist es ohne Belang ob sie als solche oder als Einzelachse betrieben wird. Die "optionalen" Komponenten in Abbildung 11 illustrieren dieses Arrangement. Eine solche Konfiguration kann sinnvollerweise zur Verwendung von linear interpolierten Achsen eingesetzt werden.

⚠VORSICHT Es ist möglich das Übersetzungsverhältnis während der Bewegung zu ändern, aber es liegt in der Verantwortung des Anwenders Übersetzungsverhältnisse zu wählen die einen sicheren Betrieb gewährleisten. Der Wechsel von einem kleinen zu einem großen Verhältnis kann eine Achse zu schnell an seine physikalische Endposition führen und zu Beschädigungen der Mechanik führen. Auch das Verletzungsrisiko kann sich aufgrund sich sehr schnell in der Geschwindigkeit ändernder Teile erhöhen.

☞ Anmerkung: Im Gegensatz zu den trapez-förmigen, S-Kurven- und Geschwindigkeits-Profilmodi, verfügt das Electronic Gearing Profil über keine explizite Erkennung ob eine Bewegung abgeschlossen wurde oder nicht. Daher wird das "motion complete" Bit des Event Status Registers, sowie das "in-motion"-Bit des Status Aktivity Registers in diesem Profil nicht funktionieren.

-o-

External Profile Mode

Die POSYS 1800-B verfügt über 64KB Dual Port RAM welches nun auf 2 Arten zugegriffen werden kann.

1. Entweder als Memory-Mapped Dual Port RAM mit fest-eingestellte Adresse, oder

2. mit E/A-Zugriff, welches aber etwa halb so schnell ausgeführt wird im Vergleich zum Memory-Mapped DPRAM

Dieser Speicher dient primär zwei Zwecken, einmal kann es als Trace Buffer verwendet werden, der andere Zweck ist es für die Speicherung und Ausführung komplexer Bewegungsprofile zu verwenden. Die Verwendung des Speichers für den Trace Buffer erlaubt die real-time Erfassung bestimmter Daten durch den Motion Prozessor und die automatische Speicherung im On-Board RAM. 4 verschiedene Variablen können aus einer Gesamtzahl von 28 ausgewählt werden die gleichzeitig erfasst werden sollen. Die Daten können zu jeder Update-Periode ermittelt werden. Längere Zeitspannen lassen sich einstellen (z.B. nur jede 10. Update-Periode).

Die Länge des Trace Buffers ist variabel und programmierbar. Entweder kann der gesamte Speicher verwendet werden, oder ein definierter Bereich. Es können bis zu 32 Bereiche eingestellt werden, die verschiedenen Zwecken dienen. Außerdem läßt sich der Speicher verwenden, um komplexe Bahnprofile abzuspeichern, die vom Motion Control Prozessor abgearbeitet werden.

-o-

Relevante On-Board RAM-Funktionen

Funktionen, die relevant für die Verwendung vom On-Board RAM sind:

-o-

Profile Mode

- GetBufferFunction
- GetBufferLength
- GetBufferReadIndex
- GetBufferStart
- GetBufferWriteIndex
- GetStopMode
- ReadBuffer
- SetBufferFunction
- SetBufferLength
- SetBufferReadIndex
- SetBufferStart
- SetBufferWriteIndex
- SetStopMode
- WriteBuffer

Die Funktion SetProfileMode(axis, mode) wurde um den Modus "On-Board RAM Mode = 4" erweitert.

Alle die obigen Funktionen und die benötigten Parameter werden ausführlich im Programmier-Handbuch beschrieben.

-o-

Trace Mode

- GetTraceCount
- GetTraceMode
- GetTracePeriod
- GetTraceStart
- GetTraceStatus
- GetTraceStop
- GetTraceVariable
- SetTraceMode
- SetTracePeriod
- SetTraceStart
- SetTraceStop
- SetTraceVariable

Diese Funktionen gelten nur für Trace-Funktionen Setup und Ausführung. Eine detaillierte Beschreibung kann im Programmierhandbuch gefunden werden.

-o-

Verwendung von On-Board RAM für External Profile Mode

Der On-Board RAM kann für die Speicherung komplexer Bewegungssequenzen verwendet werden. Die Bewegungsdaten die vom Host kalkuliert wurden, können in den Speicher geladen und dann vom Motion Prozessor ohne weitere Hostintervention ausgeführt werden. Bis zu 4 Achsen gleichzeitig können kontinuierlich in einem hoch-komplexen Pfad bewegt werden.

- Position
- Velocity
- Acceleration
- Jerk (S-curve)
- SegmentTime

Alle Daten sind im 32-Bit Format

- Positionsdaten sind immer absolut und repräsentieren die Zielposition am Ende des Segments.
- Geschwindigkeit ist in Quadraturzählungen/Update-Periode (Servozyklus)
- Beschleunigung (negativ und positiv) ist in Quadraturzählungen/Update-Periode^2
- Jerk ist in Quadraturzählungen/Update-Periode^3
- SegmentTime ist ein Vielfaches der Update-Periode (Servozyklus)

Wenn alle 4 Achsen aktiviert sind und die Update-Periode ist auf den Standardwert von 256 µSek. eingestellt, dann ist die schnellste verfügbare Update-Zeit für die Daten im Buffer die durch SegmentTime dargestellt werden ein Multiplikator von 1. Wenn ein SegmentTime Multiplikator von 0 festgestellt wird, wird der External Profile Mode für diese Achse beendet. Die anderen Achsen werden fortfahren mit der Ausführung ihrer Bewegungsprofile. Bewegungsfehler (Schleppabstandsfehler) und ein abrupter Stopp können ebenso ein Ende der Ausführung von Programmen im On-Board RAM herbeiführen.

Die Länge des SegmentTimes hängt vom Profiltyp und von der Applikation ab. Je kürzer die Zeit desto mehr Speicher wird benötigt. Lineare Bewegungsprofile können wesentlich längere SegmentTimes verwenden als zikulare Bewegungsprofile. SegmentTimes für zirkulare Bewegungsprofile benötigen aber in der Regel nie weniger als 2 Millisekunden für die präzise Ausführung von zirkularen Bewegungsprofilen.

-o-

Was bedacht werden muß

Wenn ein Bewegungsprofil im External Profile Mode ausgeführt wird, wird es normalerweise an das Ende des Buffers gelangen, dass zugeteilte Daten enthält. Falls es kein SegmentTimewert mit 0 findet, wird es an die Startposition zurückspringen und mit der Ausführung fortfahren. Dies kann eine potenziell gefährliche Situation darstellen. Falls die Endposition erheblich von der Startposition abweicht, könnte dies dazu führen, dass die Achse einen sofortigen Sprung zu dieser Position durchführt. Um den External Profile Mode zu verlassen, muss das letzte Segment einen SegmentTime Multiplikator von 0 für jede involvierte Achse enthalten. Die Achsen werden dann in ihrer Position verharren. Der ausgewählte Profilmodus (SetProfileMode(axis, mode)) ist dann immer noch aktiv (profil=4) und muss in den gewünschten Modus gesetzt werden für weitere Bewegungsausführungen.

-o-

Verfügbarer Speicher (RAM) zu klein für Programmausführung! Was tun?

Die Programmlänge überschreitet den zur Verfügung stehenden Speicher - dies stellt kein Problem dar. Teilen Sie die Bewegungssequenz in kleinere Sequenzen auf. Laden Sie diese Segmente und senden Sie zusätzliche Segmente zum On-Board RAM sobald der Speicher zur Verfügung steht. Der RAM wird in Bereiche aufgeteilt die einmal die aktuelle Sequenz ausführt und einmal die nachfolgenden Daten für die darauffolgende Ausführung bereithält. Die Funktionen GetBufferReadIndex und GetBufferWriteIndex sind Mechanismen um die momentane Leseposition und die neue Schreibposition für neue Daten zu bestimmen. Sobald die Abarbeitung der Anfangsdaten beendet ist, springt der Lesezeiger automatisch zur Startposition und fährt fort mit der Ausführung des Programmes.

Angenommen die folgenden Bedingungen liegen vor: die Gesamtanzahl Programmpositionen die im RAM benötigt werden betragen 26,350, aber nur 10,000 stehen zur Verfügung. Dann fahren Sie folgendermaßen fort:

- Laden Sie die Daten in die Position 0 bis 10,000
- Starten Sie den External Profile Mode
- Setzen Sie einen Unterbrechungspunkt, dass zu einem Ereignis passt sobald der BufferReadIndex die Position 5,000 im RAM passiert, oder prüfen Sie BufferReadIndex >= 5,000
- Beginnen Sie damit neue Daten in den RAM zu laden und beginnen Sie mit der Position BufferWriteIndex = 0 bis zu RAM Position = 4,999
- Prüfen Sie den BufferReadIndex bis er <= 10,000 ist
- Beginnen Sie damit neue Daten in den Speicher zu laden und beginnen Sie damit an der Position BufferWriteIndex = 5,000 und fahren Sie damit fort bis RAM Position = 10,000

Fahren Sie mit dieser Methode fort bis alle Daten geladen und ausgeführt worden sind. Stellen Sie sicher ein Segment anzuhängen mit einem SegmentTime Multiplikator = 0 um den External Profile Mode zu verlassen nach Beendigung der Sequenz.

Für weitere Details und Möglichkeiten bezgl. der Verwendung des On-Board Speichers, setzen Sie sich mit POSYS Motion Control GmbH & Co.KG in Verbindung.

-o-

Das SetStop Kommando

Normalerweise würde jedes der vorangegangenen Bahnprofile die entsprechenden Bahnen innerhalb der gesetzten Parameter ausführen bis die Bedingungen erfüllt wurden. Z.B. für ein Punkt-zu-Punkt Profil bedeutet dies, dass das Profil eine Achse solange in Richtung seiner Endposition versucht zu bewegen bis diese erreicht wurde und die Achse wieder gen "0" abbremst.

In manchen Fällen ist es aber notwendig die Achse manuell zu einem Halt zu bringen, z.B. aus Sicherheitsgründen. Dies kann mit zweierlei Methoden erreicht werden: abrupter Stop oder Stop mit programmierter Rampe. Um einen Stop auszuführen, verwendet man den Befehl SetStopMode. Um die gegenwärtige Einstellung für einen Stop abzufragen verwendet man das Kommando GetStopMode.

AbruptStop stoppt die entsprechende Achse ohne Rampe was einem Notfallstop gleichkommt. Dabei wird die Geschwindigkeit der Achse sofort auf "0" gesetzt.

SmoothStop dagegen stoppt die entsprechende Achse mit dem jeweilig eingestellten Wert für die negative Beschleunigung. Ist dieser Wert = 0 wird dann der Wert für die positive Beschleunigung verwendet.

In beiden Modi wird die Zielgeschwindigkeit auf Null (0) gesetzt. Um eine neue Bewegung zu initiieren ist es notwendig die Geschwindigkeit mit dem Kommando SetVelocity neu zu programmieren.

AbruptStop muss mit Vorsicht verwendet werden. Plötzliches Abbremsen aus einer hohen Geschwindigkeit kann die Mechanik beschädigen oder zu Verletzungen führen.

AbruptStop funktioniert in allen Profilen. SmoothStop funktioniert in allen Profilen, außer Electronic-Gear.

-o-

Aktivieren und Deaktivieren von Trajectory Generator Modules

Es gibt eine Reihe von Gründen warum es sinnvoll sein kann das Trajectory Generator Module zu aktivieren oder deaktivieren. Weitere Informationen sind im Kapitel „Control Flow Overview“ nachzulesen. Zusätzlich zur manuellen Deaktivierung verschiedener Kontrollmodule gibt es eine Reihe Bedingungen unter denen die automatische Deaktivierung sinnvoll sein kann, wie z.B. Ereignis-bedingte oder durch Breakpoints bestimmte Bedingungen. Im Kapitel “SetEventAction Processing,” sind weitere Informationen erhältlich.

Falls der Trajectory Generator Module deaktiviert ist, wird die Sollposition seinen gegenwärtigen Wert beibehalten. Alle anderen Profile und Kommandos werden ignoriert. Ergänzend, falls der Position Loop aktiviert ist, wird zu dem Zeitpunkt bei dem der Trajectory Generator Module deaktiviert wird der Schleppabstandsfehler auf Null (0) gesetzt (entspricht einem ClearPositionError Kommando).

Ein vorher deaktivierter Trajectory Generator Module kann auf verschiedene Arten wieder aktiviert werden. Falls das Kommando SetOperatingMode verwendet wurde um es zu deaktivieren, kann das gleiche Kommando verendet werden um es wieder zu aktivieren. Falls der Trajectory Generator Module deaktiviert wurde aufgrund einer automatischen, Ereignis-bedingten Aktion (siehe „SetEventAction Processing“ für weitere Informationen), muss das Kommando RestoreOperatingMode ausgeführt werden um das Trajectory Generator Module wieder zu aktivieren.

-o-

Position Loop

Überblick

Für die Achsen der POSYS® die im Servomodus betrieben werden, wird die Regelschleife als Teil der Grundmethode benutzt, den Drehmomentsollwert zu bestimmen. Der Grundgedanke der Regelschleife ist die Istposition so nahe wie möglich mit der Sollposition gleichzusetzen und entsprechende Angleichungen durch Anpassung des Drehmomentsollwertsignals dahingehend vorzunehmen, dass entsprechend den Parametervorgaben die Soll- und Istposition möglichst identisch sind.

Um dies zu bewerkstelligen wird der Ausgangswert mit der aktuellen Enkoderposition verglichen um den Schleppabstandsfehler zu errechnen, welches wiederum durch die Regelschleife geschickt wird. Das skalierte Ergebnis der Filterkalkulation ist die Drehmomentsollwertvorgabe, welches als impulsbreiten moduliertes Signal mit 10-Bit Auflösung und 20 kHz, 8-Bit Auflösung mit 80 Khz oder als 16-Bit Wert an einen D/A-Konverter geschickt wird um ein ±10V analoges Ausgangssignal zu generieren.

Die übergeordnete Regelschleife ist in zwei Hauptteile gesplittet – Die PID-Regelschleife und die Biquadfilterung. Die Regelschleife generiert die initiale Sollwertvorgabe, während der duale Biquadfilter verwendet werden kann verschiedene Frequenzfilterungen wie z.B. Notch, Lowpass und Bandpass auszuführen. Nachdem die Ausgabe der Regelschleife und der Biquadfilterung abgeschlossen wurde, kann es weiteren Begrenzungen unterzogen werden um innerhalb eines bestimmten Bereiches zu verbleiben um Treibern/Verstärkern, Motoren oder physikalischen Gegebenheiten gerecht zu werden wie in der nächsten Abbildung dargestellt.

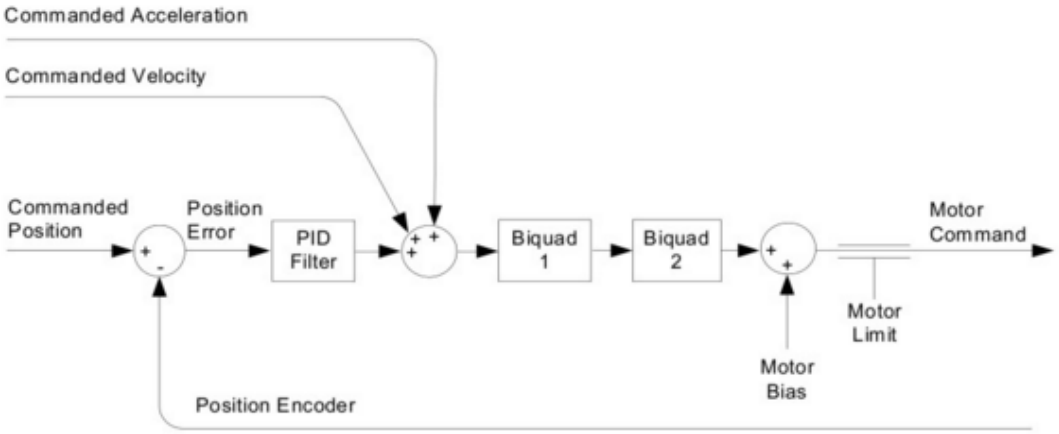

Abbildung 12: Regelschleife und Biquadfilterung

Um eine gute Lageregelung zu ermöglichen, erfordern alle Servo-Anwendungen, dass sichere und stabile PID-Parameter angegeben werden. Die Nutzung der Dual Biquad Filter, auf der anderen Seite, ist optional und hängt von der Art und Komplexität der Steuerungsaufgabe ab. Je anspruchsvoller die Anwendung, desto wahrscheinlicher ist, dass die Biquads nützlich sein werden.

-o-

PID Loop

Der mit der POSYS® benutzte Servofilter ist ein PID-Algorithmus mit "velocity und acceleration feedforward" Termini und einem Ausgangsmaßstabsfaktor. Ein Integrationslimit sichert nach oben ab gegen einen akkumulierten Fehler. Ein optionaler "bias" kann zur Filterkalkulation addiert werden, um die letzte Motorausgabeanweisung zu erzeugen. Ein Grenzwert für den Filterausgang sorgt für eine zusätzliche Beschränkung. Abbildung 13 gibt einen Überblick über den PID loop:

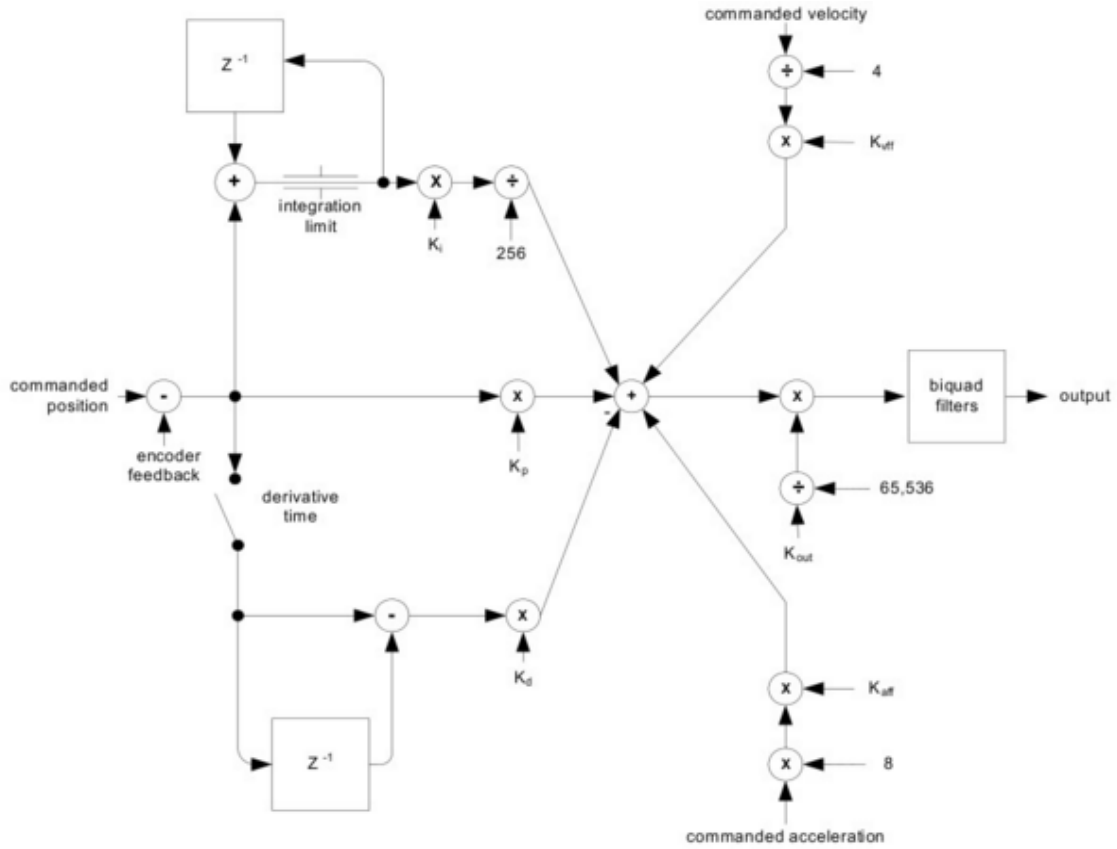

Abbildung 13: Struktur des digitalen Filters

Die PID+Vff+Aff Formel, zusammen mit dem Skalierungsfaktor und den BIAS termini, ist wie folgt:

$$Output_n = \left[K_p E_n + K_d(E_k - E_{k-1}) + \sum_{j=0}^{n} E_j \times \frac{K_i}{256} + K_{vff}\left(\frac{CmdVel}{4}\right) + K_{aff}(CmdAccel \times 8) \right] \times \frac{K_{out}}{65,536}$$

wobei

k	= n – modulus (n/derivative time)
En	= position loop error at the derivative sampling interval (Commanded Position – Actual Position)
Ek	= position loop error at the derivative sampling interval
Ki	= Integral Gain
Kd	= Derivative Gain
Kp	= Proportional Gain
Kaff	= Acceleration feed-forward
Kvff	= Velocity feed-forward
Kout	= scale factor for the output command

Tabelle 11: PID Loop Parameter

Alle Filterparameter, die Drehmomentsignalbegrenzung, und der "Motor Bias" sind programmierbar, so dass der Filter durch das Anwenderprogramm fein abgestimmt werden kann. Die einstellbaren Bereiche für die Parameter, Formate und Interpretationen werden in der folgenden Tabelle gezeigt:

Terminus	Name	Bereich
Ilimit	Integration Limit	Unsigned 32 bits (0 to 2,147,483,647)
Ki	Integral Gain	Unsigned 16 bits (0 to 32,767)
Kd	Derivative Gain	Unsigned 16 bits (0 to 32,767)
Kp	Proportional Gain	Unsigned 16 bits (0 to 32,767)
Kaff	Acceleration feed-forward	Unsigned 16 bits (0 to 32,767)
Kvff	Deceleration feed-forward	Unsigned 16 bits (0 to 32,767)

Terminus	Name	Bereich
Kout	Output scale factor	Unsigned 16 bits (0 to 32,767)
Derivative Time	Derivative Sampling Time	Unsigned 16 bits (0 to 32,767)

Tabelle 12: PID Loop Parameter Values

Um PID-Regelparameter zu setzen, verwenden Sie das Kommando SetPositionLoop. Um diese zurückzulesen verwenden Sie das Kommando GetPositionLoop.

-o-

Integration Limit

Der Integralbegrenzer wird verwendet, um eine Grenze für den absoluten Wert zu setzen, welches beiträgt zur PID-Ausgabe durch den Integralfilter. Sein Standardwert nach einem Reset ist Null (0), welches dazu beiträgt, dass auch der Integralwert der PID-Filterausgabe gleich Null (0) ist. Um Ki zu verwenden muss dieser > Null (0) sein. Für weitere Informationen bzgl. der Skalierung des Integralwertes verweisen wir auf das Programmierhandbuch. Das Kommando zum setzen des Integralwertes lautet SetPositionLoop, zum lesen des eingestellten Wertes verwenden Sie das Kommando GetPositionLoop.

-o-

Output Scaling

Der Kout Parameter kann dazu verwendet werden die PID-Regelparamter, in Situationen in denen dies notwendig erscheint, herunter zu skalieren. Dies wird dadurch bewerkstelligt indem das Filterergebnis mit Kout/65536 multipliziert wird. Der Effekt ist, dass der Bereich für Kp erweitert wird, welches typischerweise innerhalb eines Bereiches von 1 bis 150 liegt wenn keine Ausgangsskalierung vorgenommen wird. Der Kout-Wert wird mit dem Kommando SetPositionLoop gesetzt. Mit GetPositionLoop und dem entsprechenden Parameter wird der programmierte Wert gelesen.

Anders als bei den meisten anderen PID-Parametern deren Standardwert = Null (0) ist, ist der Standardwert für Kout nach einem Reset oder nach dem Einschalten 65535 bzw. 100%.

-o-

Derivative Sampling Time

Unter normalen Umständen wird der Derivative Term der PID-Regelschleife zu jedem Zyklus neu berechnet. Unter bestimmten Voraussetzungen kann es aber wünschenswert sein die Kalkulation nur jede 5. oder 10. Schleife neu kalkulieren zu lassen um Systemstabilität zu erhöhen oder das Tuning des Systems zu erleichtern. Das Kommando SetPositionLoop setzt diesen Wert. Das Kommando GetPositionLoop liest den programmierten Wert zurück.

Der spezifizierte Wert ist die Anzahl Servozyklen pro Motion Controller Servoloop-Updaterate. z.B. falls die Servoloop-Updaterate auf 200 µsec gesetzt wurde mit dem Kommando SetSampleTime (effektiv: 204,8 µsec), dann wird ein Wert von „1“ programmiert im Derivative Time Register. Ein Wert von „10“ bedeutet dass alle 10 Servozyklen (effektiv: 2048 µsec, oder 2,048 msec) eine Neukalkulation stattfindet.

Eine Änderung des Derivativ Sample Time hat keinen Effekt auf die übergeordnete Motion Controller Sample Time welches mit dem Kommando SetSampleTime eingestellt wird.

Der Standardwert für Derivative Sampling Time nach dem Einschalten oder einem Reset ist „1“.

-o-

Dual Encoder Support

Dual Encoder Support

Die Multi-Achsen Versionen der POSYS® 18xx-B unterstützen Dual-Enkoder-Rückmeldung. Dies ist besonders dann von Nutzen wenn die tatsächliche Position z.B. aufgrund von Backlash von der gemeldeten Position abweicht. In dieser Konfiguration ist die Enkoderrückmeldung einer 2. Achse verknüpft mit der zu kontrollierenden Achse als „Derivative Term“. Wenn der Enkodereingang der 2. Achse in dieser Weise verwendet wird, steht sie für einen 2. Motor als Rückmeldeeingang nicht mehr zur Verfügung. Dieses Funktion ist auch nicht verfügbar für Karten mit erweiterten PID-Regelparametersätzen (POSYS 1800-B Piezo).

Der Auxiliary-Encoder wird für die Sinuskommutierung verwendet bei den Achsen mit EC-Servomotoren und aktiviertem Dual-Enkoder-Modus.

Die folgende Abbildung gibt einen Überblick über die Funktionsweise des Dual Enkoder Modus.

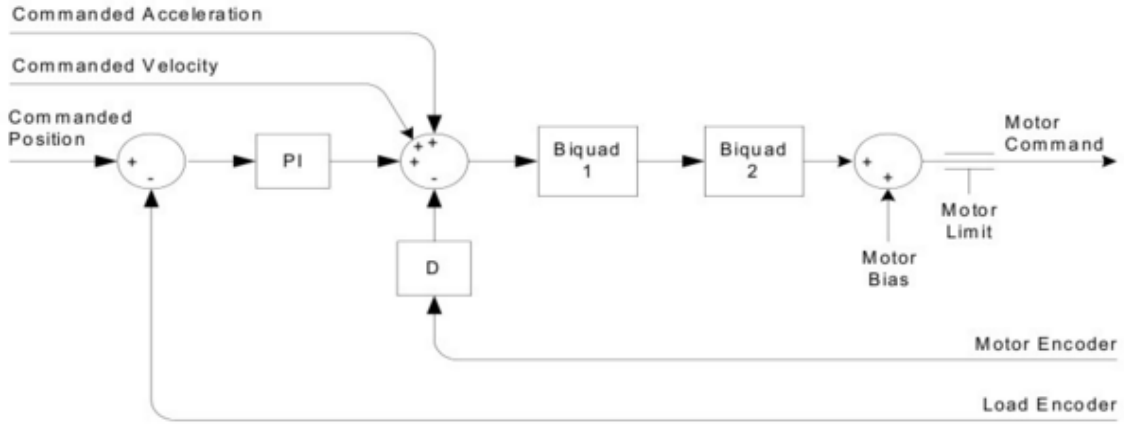

Abbildung 14: Motion Control Dual Loop Flow

-o-

Dual Encoder PID Loop Algorithmus

Die Formel die für den Servofilter Anwendung findet in Verbindung mit dem Dual-Enkoder Algorithmus weicht von der Formel für die normale Verwendung des Servofilters in folgender Weise ab:

$$Output_n = \left[K_p E1_n - K_d(P_n - P_{n-1}) + \sum_{j=0}^{n} E1_j \times \frac{K_i}{256} + K_{vff}\left(\frac{CmdVel}{4}\right) + K_{aff}(CmdAccel \times 8) \right] \times \frac{K_{out}}{65{,}536}$$

wobei

EIn	= sind die akkumulierten Fehler vom 1. Enkoder
Pn	= Ist die Position des 2. Enkoders
Ki	= Integral Gain
Kd	= Derivative Gain
Kp	= Proportional Gain
Kaff	= Acceleration feed-forward
Kvff	= Velocity feed-forward
Kout	= scale factor für das Drehmomentsollwertsignal

Tabelle 13: Dual Encoder PID Loop Algorithmus

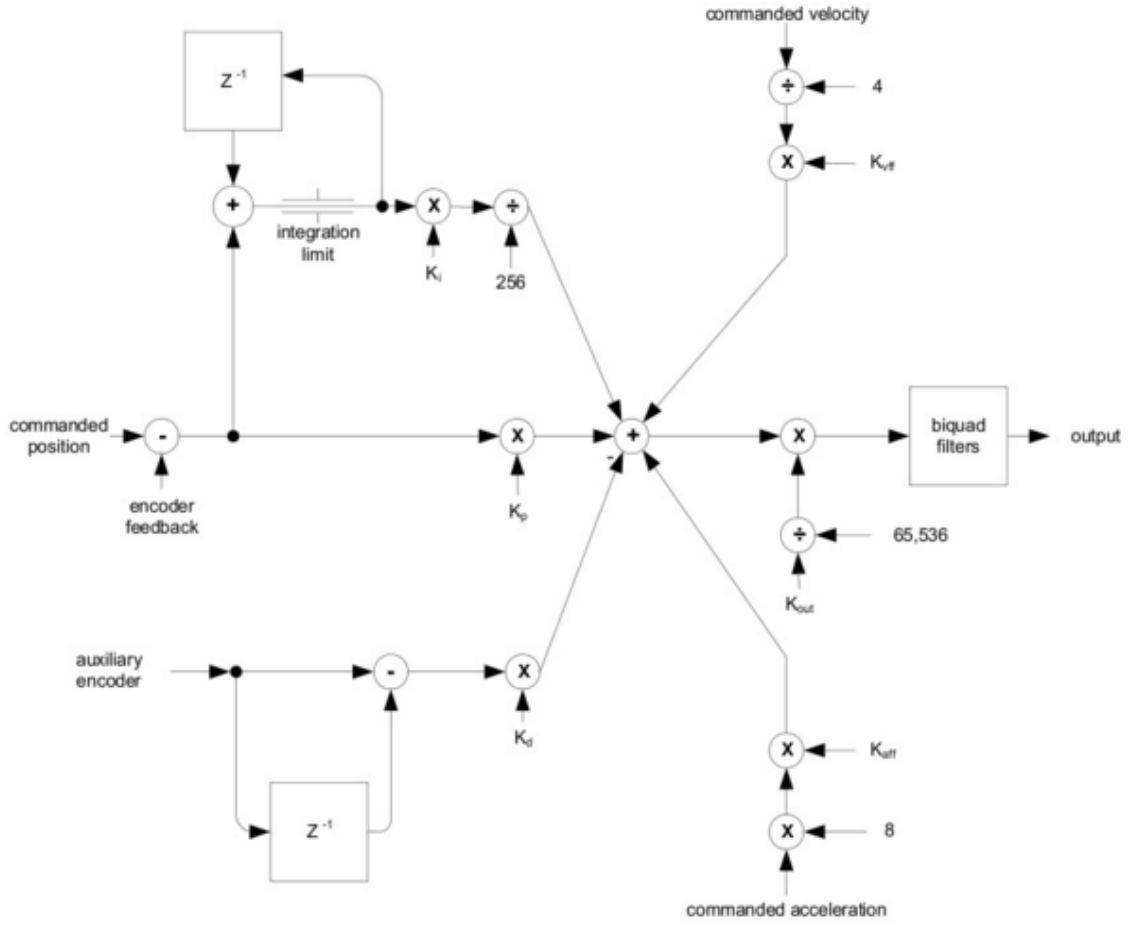

Abbildung 15: Motion Control Dual Loop Digital Filter

-o-

Konfiguration von Dual-Enkoder Unterstützung

Das Kommando SetAuxiliaryEncoderSource wird verwendet um die Dual-Enkoder Unterstützung zu aktivieren. Der auxiliaryAxis Parameter bestimmt welcher Achsenenkodereingang verwendet wird um die primäre Achse zu unterstützen. Der mode Parameter aktiviert bzw. deaktiviert den Dual-Enkoder-Modus. Solange der Dual-Enkoder-Modus deaktiviert ist, funktioniert der Servofilter wie unter PID Schleifenalgorithmus beschrieben.

Falls die Anwendung die aktuelle Position des Auxiliary Enkoders bestimmen muss, dann verwenden Sie das Kommando GetActualPosition mit Achsenspezifizierung des Auxiliary Enkoders.

Der Auxiliary-Enkoder sollte eine Auflösung aufweisen die mindestens genauso fein oder feiner ist als die des Hauptenkoders. Dies wird helfen potenziell instabile Systemkonditionen zu vemeiden.

Für POSYS 1800-B Karten gilt folgendes: Die Anzahl Enkodereingänge ist gleich der Anzahl unterstützter Achsen. In anderen Worten: eine 4-Achsen Karte hat 4 Anschlüsse für Enkoder. Wird ein Enkodereingang für Dualloop verwendet, reduziert sich die Gesamtzahl unterstützter Achsen auch um eins.

-o-

Biquad Output Filter

Biquad Output Filter

Ein Biquad ist ein digital strukturierter Filter. Mit den richtigen Koeffizienten kann es wie ein Low-Pass-Filter, High-Pass-Filter, BandPass-Filter, Notch-Filter, oder benutzerdefinierter Filter programmiert werden. Programme wie Octave (www.octave.org) können verwendet werden, um die Koeffizienten zu ermitteln.

Die POSYS 1800-B unterstützt zwei programmierbare Biquad Ausgangsfilter für jede Achse. Diese Filter sind verkettet. Wenn beide aktiviert sind, speist die Ausgabe von Filter1 die Eingabe von Filter2. Wenn Filter1 deaktiviert ist (Vorgabe), wird der gesamte Filter umgangen, und die Motorsollwertvorgabe geht ungefiltert auf den Treiber/Motor.

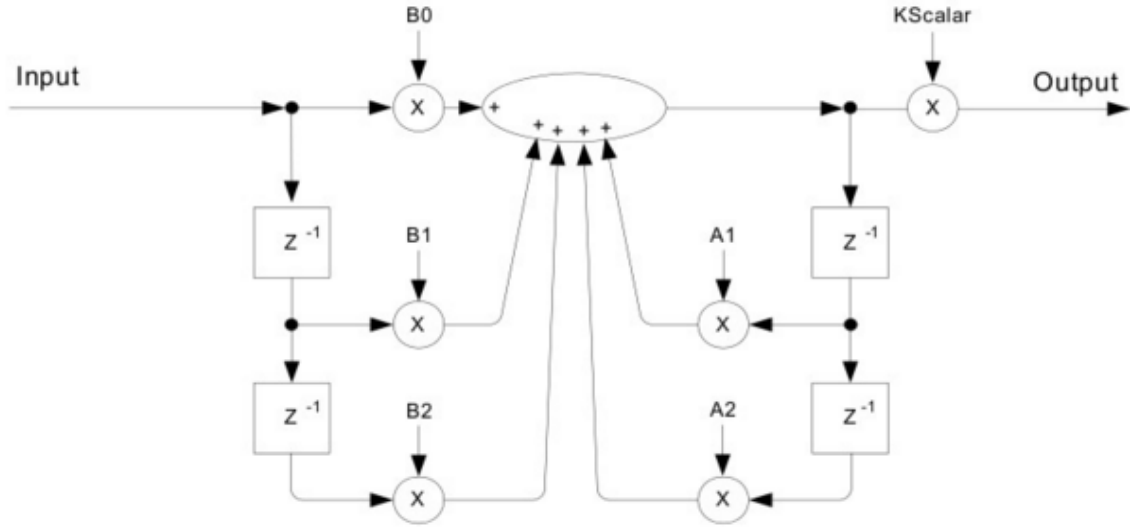

Abbildung 16: Biquad Algorithm Flow

Der Ausgang des Filters zum Zeitpunkt n ist mit der folgenden Gleichung bestimmt:

$$Y_n = K \times (B_0 \times X_n + B_1 \times X_{n-1} + B_2 \times X_{n-2} + A_1 \times Y_{n-1} \times A_2 \times Y_{n-2})$$

wobei

Yn	= Ausgabe des Filters zum Zeitpunkt n
Xn	= Eingabe zum Filter zum Zeitpunkt n
K	= positiver Skalierungswert zur Vermeidung von Rundungsfehlern
B0	= programmierbarer Biquad Koeffizient
B1	= programmierbarer Biquad Koeffizient
B2	= programmierbarer Biquad Koeffizient
A1	= programmierbarer Biquad Koeffizient
A2	= programmierbarer Biquad Koeffizient

Tabelle 14: Biquad Koeffizientenparameter

Die folgende Tabelle zeigt Filterkoeffizienten welche über das Kommando SetPositionLoop gesetzt wurden. Die gesetzten Einstellungen können mit dem Kommando GetPositionLoop ermittelt werden. Die folgende Tabelle fasst die Darstellung und den Bereich der einstellbaren Biquad-Parameter zusammen.

Term	Name	Art & Bereich
K	Biquad scalar	unsigned 16 bits (0 to 32,767)
A1	Koeffizient A1	signed 16 bits (–32,768 to 32,767)
A2	Koeffizient A2	signed 16 bits (–32,768 to 32,767)
B0	Koeffizient B0	signed 16 bits (–32,768 to 32,767)

Term	Name	Art & Bereich
B1	Koeffizient B1	signed 16 bits (–32,768 to 32,767)
B2	Koeffizient B2	signed 16 bits (–32,768 to 32,767)

Tabelle 15: Parameterwerte für Biquadkoeffizienten

-o-

Biquadkoeffizienten bestimmen

Typischerweise sind Koeffizienten, die in Bi-Quad Filtergleichungen verwendet werden, kleine Fließkommazahlen. Um daher Rundungsfehler zu vermeiden wenn diese Zahlen als 16-Bit Werte gespeichert werden, wird der K-Koeffizient mit 227 skaliert um die Eingabe der anderen Koeffizienten als Integer zu erlauben.

Z.B. in Octave sind die Koeffizienten für einen zweit-rangigen Butterworth-Filter folgendermaßen:

```
[b,a] = butter(2,0.1)
```

Die Ergebnisse in den Koeffizienten:

```
b0 = 0.020083
b1 = 0.040167
b2 = 0.020083
a1 = -1.56102
a2 = 0.64135
```

Wenn nun die Filtergleichung der POSYS® mit der Filtergleichung von Octave verglichen wird, bemerkt man einen kleinen Unterschied dahingehend, dass die ax Komponenten in Octave subtrahiert werden, wohingegen sie in der POSYS® addiert werden. Das führt dazu, dass die a1 und a2 Koeffizienten von Octave (oder Matlab) mit -1 multipliziert werden müssen, ehe sie an die POSYS® gesendet werden können.

Das Ergebnis sieht dann folgendermaßen aus:

```
b0 = 0.020083
b1 = 0.040167
b2 = 0.020083
a1 = 1.56102
a2 = -0.64135
```

Diese Werte werden skaliert und als Ausgangsfilterkoeffizienten gesetzt, sobald der Ausgangsskalierungsfaktor K bestimmt wurde.

-o-

Bestimmung des Bi-Quad Skalierungsfaktors

Um höchste Ausgabegenauigkeit zu erlangen, sollte der programmierbare Skalierungsfaktor K so gewählt werden, dass er mit dem größten absoluten Wert der dem Wert 32767 (größte, positive Wert für ein 16-Bit Integer mit Vorzeichen) am nächsten ist, aus dem Satz von Koeffizienten (in diesem Fall a1) skaliert werden kann. Dann kann K folgendermaßen bestimmt werden:

$$a1 = K * 32767 * 2^{-27}$$

Umstellung der Gleichung:

Nach Umstellung: $K = (a1 * 2^{27}) / 32767$

Einsetzen der Werte: $K = (1.56102 * 2^{27}) / 32767$

Ergebnis: K = 6394

-o-

Skalierung der Bi-Quad Koeffizienten

Nachdem der optimale K-Skalierungsfaktor ermittelt wurde, können die äquivalenten „Integer“ der Bi-Quad Koeffizienten berechnet werden. Die Integer-Werte (B0, B1, B2, A1 und A2) können wie im folgenden Beispiel errechnet werden:

$$b0 = B0 * K * 2^{-27}$$

Umstellung der Gleichung:

Nach Umstellung: $B0 = (b0 * 2^{27}) / K$

Einsetzen der Werte: $B0 = (0.020083 * 2^{27}) / 6394$

Ergebnis: $B0 = 422$

Bei der Verwendung dieser Formel ($X = (x * 2^{27}) / K$) auf alle Koeffizienten werden folgende Werte ermittelt:

B0 = 422
B1 = 843
B2 = 422
A1 = 32767
A2 = -13463
K = 6394

-o-

Output Limit

Output Limit

Die Motorausgabebegrenzung hält die Filterausgabe davon ab, eine bestimmte Spannungsvorgabe für jede Richtung an den Verstärker auszugeben bzw. zu überschreiten. Wenn der Filter einen Wert, größer als die Grenze, erzeugt, nimmt der Motorbefehl die programmierten Grenzwerte. Mit SetMotorLimit wird der Grenzwert programmiert. Mit GetMotorLimit wird der programmierte Wert gelesen. Der spezifizierte Wert ist eine 16-Bit vorzeichenlose Zahl von 0 bis 32767. Der spezifizierte Wert ist der größtmögliche Wert, der an den Verstärker/Motor ausgegeben wird. Zum Beispiel wenn die Motorgrenze auf 30000 gesetzt wurde (oder 91.6 % es maximalwertes), werden Motorausgabewerte die größer als 30000 wären, dann als 30000 ausgegeben, und Motorausgabewerte weniger als -30000 werden als -30000 ausgegeben.

Ein Output Limit kann nur im Closed Loop Mode programmiert werde, da im Open Loop Mode die Register direkt beschrieben werden.

Der Standardwert der nach einem Reset bzw. PowerOn aktiv ist, ist 100% bzw. 32767.

-o-

Motor Bias

Motor Bias

Wenn eine Achse abhängig ist von einer netto-bedingten von aussen wirkenden Kraft die in einer Richtung zieht/drückt (wie eine senkrechte abwärts von der Schwerkraft gezogenen Achse), kann der Servofilter eine Kompensation vornehmen, indem er eine ständige Vorgabespannung an den Filter schickt die beiträgt der Kraft entgegen zu wirken. Das Kommando hierfür heisst SetMotorBias. Der programmierte Wert kann mit GetMotorBias gelesen werden.

Solange der Regelkreis geschlossen ist, wirkt der MotorBias zu allen Zeiten. Falls der Position Loop deaktiviert ist, aber der Bahngenerator weiterhin aktiv ist, wirkt auch in diesem Fall der MotorBias.

Wenn die Regelschleife und der Bahngenerator deaktiviert werden, wird der MotorBias dann nur nach einem Übergang auf diesen Status angewandt, und eine nachfolgende Änderung des Motor-Befehls-Registers wird ohne MotorBias angewandt. Für weitere Informationen sehen Sie bitte auch im Kapitel "Aktivieren und Deaktivieren von Kontrollmodulen" nach.

Zum Beispiel, falls ein MotorBias mit einem Wert von +1000 (=~+3%) gesetzt wurde, in dem Moment in dem das Kommando SetOperatingMode ausgeführt wird, um den Bahngenerator und die Regelschleife zu deaktivieren, oder zu dem Zeitpunkt bei dem eine sicherheitsrelevante Aktion ausgeführt wird, wie die Erkennung eines Schleppabstandsfehlers und diese Module sind automatisch deaktiviert, wird die Vorgabespannung von +1000 erhalten bleiben. Falls der Anwender bei dem Versuch diesen Prozess zu bereinigen einen neuen Vorgabewert von +2000 vorgibt, wird dieser Wert unabhängig vom MotorBiaswert ausgegeben.

Der Standardwert ist null (0).

Falls der spezifizierte Wert für eine Motorvorgabespannung nicht korrekt gesetzt wurde um eine externe Kraft zu kompensieren, kann es sein dass die Achse in die eine oder andere Richtung plötzlich davonläuft, nachdem das Kommando SetOperatingMode ausgeführt wurde. Es liegt in der Verantwortung des Anwenders einen MotorBiaswert zu wählen der eine sichere Operation gewährleistet.

-o-

Deaktivieren und Aktivieren des Position Loop Moduls

Deaktivieren und Aktivieren des Position Loop Moduls

Es gibt verschiedene Gründe warum es wünschenswert sein kann das Position Loop Modul zu deaktivieren (siehe Aktivieren und Deaktivieren von Kontrollmodulen). Zusätzlich exisiteren Ereignis-bedingte Aktionen die als Ergebnis die Deaktivierung des Position Loop Modules zur Folge haben (siehe SetEventAction Processing).

Falls das Position Loop Modul deaktiviert wurde, kann das auf den allgemeinen Kontrollfluss zweierlei Auswirkungen haben, abhängig davon ob auch der Bahngenerator aktiv oder deaktiv ist.

Bahngenerator deaktiviert: Falls der Bahngenerator deaktiviert ist, wenn die Regelschleife deaktiviert wird, wird die Ausgabe des Lagereglermodules ein 16-Bit Wort sein, der von einem programmierbaren Motorbefehlsregister abgeleitet, mit dem Kommando SetMotorCommand programmiert wurde und mit dem Kommando GetMotorCommand gelesen werden kann.

Bahngenerator aktiviert: Falls der Bahngenerator aktiviert ist, wenn die Regelschleife deaktiviert wird, wird das Lagereglermodul dann umgangen, und der Ausgangswert des Bahngenerators wird der Motorbefehlswert sein.

Egal ob das Lagereglermodul aktiviert oder deaktiviert ist, der eigentliche wirksame Motorbefehlswert kann mit dem Kommando GetActiveMotorCommand abgefragt werden. Dies ist der in dem Moment tatsächlich an die nachfolgenden Module und den Verstärker übergebene Wert für die Motorausgabe.

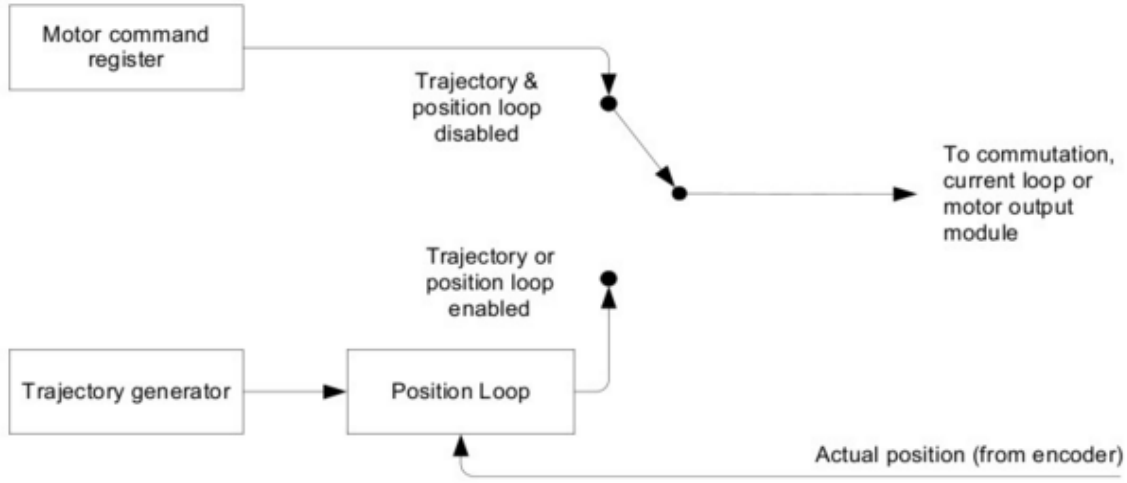
Motor command register
Trajectory & position loop disabled
To commutation, current loop or motor output module
Trajectory or position loop enabled
Trajectory generator
Position Loop
Actual position (from encoder)

Abbildung 17: Motor Control Pfade, Bahngenerator aktiviert/deaktiviert

Ein vorher deaktiviertes Lagereglermodul kann auf verschiedene Arten wieder freigegeben werden. Falls das Modul mit dem Kommando SetOperatingMode daktiviert wurde, kann ein neuerlicher SetOperatingMode Kommando es wieder aktivieren. Wenn das Lagereglermodul als Teil einer automatischen Ereignis-bezogenen Aktion (siehe SetEventAction Processing, für Details) deaktiviert wurde, verwendet man dann das Kommando RestoreOperatingMode um das Modul wieder zu aktivieren.

Ungeachtet wie das Modul wieder freigegeben wird, zu dem Zeitpunkt zu dem die Reaktivierung initiiert wird, werden bestimmte Vorgänge ausgeführt um unerwartetes Verhalten der Achse zu vermeiden. Im besonderen werden alle Lagereglerzustandsgrößen auf Null (0) gesetzt.

-o-

Position Loop Werte lesen

Es gibt eine Reihe Kommandos die verwendet werden können, verschiedene Werte für das Position Loop auszulesen. Unter anderem sind dies GetCommandedPosition um die Sollposition zu lesen, GetActualPosition um die tatsächliche aktuelle Position zu lesen die vom Enkoder zurückgeliefert wird, GetPositionError um den Schleppabstandsfehler (die Differenz der beiden vorausgegangenen Werte) zu erfahren und GetActiveMotorCommand um die aktuelle Motorausgabe des Position Loop Modules zu lesen. Diese Werte sind nicht nur lesbar sondern auch für das Tracing auswählbar.

Neben diesen Registern, um die Abstimmung weiter zu erleichtern, exisitieren eine Anzahl weiterer Lagereglerwerte, die zurückgelesen als auch nachverfolgt werden können. Das Kommando zum zurücklesen lautet GetPositionLoopValue.

Die Variablen innerhalb des Position Loops sind in der nachfolgenden Tabelle gelistet und zusammengefaßt.

Variablenname	Funktion
Integrator Sum	Dieses Register liefert die Summe des Integrator's für den Position Loop.
Integral Contribution	Dieses Register hält den umfassenden Beitrag vom Integrator zum Position-PID-Loop.
Derivative	Dieses Register hält den Lagefehlerableitungswert. Das heißt der Unterschied zwischen dem gegenwärtigen Lagefehler und dem vorhergehenden Lagefehler.
BiQuad1 Input	Dieses Register beinhaltet den Eingabewert für das BiQuadfilter #1.
BiQuad2 Input	Dieses Register beinhaltet den Eingabewert für das BiQuadfilter #2.

Tabelle 16: Position Loop Werte

-o-

Parameter Update and Breakpoints

Parameter Buffering

Verschiedene Parameter müssen für die POSYS® angegeben werden, damit eine Achse richtig kontrolliert wird. In einigen Fällen kann es wünschenswert sein, dass einige Parameter zur exakt gleichen Zeit aktiv/aktuell werden um exakt synchronisierte Bewegungsabläufe zu gewährleisten. Um diese Arten von Profilparametern und einige andere Arten von Parametern wie Servoparameter zu unterstützen (POSYS® Achsen in Servomodus) werden sie in Buffer geladen. Diese zwischengespeicherten Befehle werden in einen Bereich der POSYS® geladen, der die eigentliche POSYS®-Verhaltensweise nicht beeinflusst, bis ein besonderes Ereignis, bekannt als ein Update, eintritt. Ein Update veranlasst die zwischengespeicherten Register in die aktiven Register zu laden, die POSYS® dazu veranlassend die neuen Parameter zu verwenden.

Es gibt drei verschiedene Arten von gebufferten Registern, von denen jeder unabhängig aktualisiert und aktiv gemacht werden kann. Es sind die Profilparameter, die Position Loop Parameter und die Regelschleifenparameter. Das Kommando SetUpdateMask steuert, welche dieser drei Arten nach Erhalt eines Update-Kommandos in die aktiven Register kopiert werden. Der Befehl GetUpdateMask dient zum Lesen des spezifizierten Maskenwertes. Getrennt aktualisierbare Parameter sind sinnvoll, weil sie es ermöglichen (zum Beispiel) Profilparameter On-the-fly zu ändern, ohne die Position Loop Parameter oder die Regelschleifenparameter zu modifizieren, oder die Regelschleifenparameter zu ändern ohne Position Loop Parameter oder Profilparameter zu ändern. Viele Anwendungen werden diese Vielfalt nicht unbedingt benötigen, jedoch ist es nicht ungewöhnlich diese Maske einmal zu setzen so dass sämtliche gebufferten Register immer aktualisiert werden.

Die folgende Kommandosequenz zeigt das Laden und Aktualisieren einer Reihe von Parametern. In diesem Fall sind es ein neuer Profil-Modus, Position, Geschwindigkeit und Beschleunigung (alle Kommandos sind gebuffert) die geladen werden, gefolgt von einem Update-Befehl, um sie zu aktivieren.

SetProfileMode Axis1, trapezoidal	setze Profilmodus trapezförmig für Achse 1
SetPosition Axis1, 12345	lade Endposition für Achse 1
SetVelocity Axis1, 223344	lade Geschwindigkeit für Achse 1
SetAcceleration Axis1, 1000	lade Beschleunigung für Achse 1
SetUpdateMask Axis1, Profile	angeben, dass nur eine Aktualisierung der Profile-Parameter erfolgen soll
Update	Doppelt-gepufferte Register werden in die aktiven Register kopiert, und leiten damit die Bewegung ein

Tabelle 17: Beispiel für Parameter Buffering

Nachdem diese Sequenz abgeschlossen wurde, werden die zwischengespeicherten Register für diese Parameter (einschließlich des Profilmodus selbst) in die Register der POSYS® geladen, aber das Bahnprofil läuft noch nach den vorhergehenden Werten ab. Nur wenn ein Update-Kommando ausgeführt wird, wird der Profilmodus eigentlich zu trapezförmig und die angegebenen Parameter in den Trajektoriegenerator geladen die die POSYS® sofort veranlassen, das angegebene Programm zu starten.

Der mit dem Kommando SetUpdateMask eingestellte Wert ist nur durch eine spätere Ausführung des Kommandos SetUpdateMask erneut veränderbart. Das heißt, sein Wert selbst wird nicht verändert durch ein Update -Kommando oder durch andere Kommandos.

Der Standardwert des Update-Maske für POSYS 1800-B ist Profil-Parameter und Regelschleifenparameter zu aktualisieren.

-o-

Updates

Es gibt drei unterschiedliche Methoden für ein Update. Sie werden nachfolgend aufgelistet:

1. Update Kommando—Die einfachste Art ist, ein Update-Kommando zu geben. Dies veranlasst die Parameter für die programmierte Achse sofort zu aktualisieren.

2. MultiUpdate Kommando—Der multiple Achsenupdate, dass bei der Verwendung des MultiUpdate-Befehles

angegeben wird, veranlasst, dass mehrere Achsen gleichzeitig aktualisiert werden. Dies kann nützlich sein, wenn interpolierende Achsen synchronisiert werden sollen. Dieses Kommando nimmt ein 1-Wort Argument aus einer Bitmaske, welches aus einem Bit besteht und jeder Achse zugewiesen ist. Dieser Befehl hat bei Ausführung die gleiche Wirkung wie das Update-Kommando, welches an jede Achse einzeln geschickt wird.

3. Breakpoints—Es gibt eine sehr nützliche Funktion die von der POSYS® unterstützt wird, die programmiert werden kann, ein Update-Befehl automatisch zu generieren - wenn ein vorprogrammierter Zustand wahr wird. Diese Möglichkeit wird Unterbrechungspunkt genannt, und es ist nützlich für Operationen wie z.B.: "automatisch die Geschwindigkeit verändern wenn eine besondere Position erreicht wird", oder "die Achse abrupt stoppen wenn ein bestimmtes externes Signal aktiv wird". Jedoch Im Gegensatz zum Standard Update Kommando, kontrolliert das Kommando SetBreakpointUpdateMask den Transfer der zwischengespeicherten Kommandos zu den aktiven Registern.

Gleich welche Update-Methode benutzt wird, zu dem Zeitpunkt zu dem der Update passiert, werden alle zwischengespeicherten Register zu den aktiven Registern kopiert. Umgekehrt, bevor das Update passiert, haben zwischengespeicherte Register oder die Ausführung zwischengespeicherter Befehle so lange keinen Effekt auf das laufende System bis das Update-Kommando tatsächlich ausgeführt wurde.

Im Gegensatz zur Profilerstellung sind die meisten Parameterkommandos nicht gebuffert und einige sind gebuffert. Die folgende Liste zeigt alle gebufferten Kommandos.

Doppelt gebufferte Kommandos

Trajectory	Position Loop	Current Loop
SetProfileMode ClearPositionError SetStopMode SetPosition SetVelocity* SetAcceleration SetDeceleration SetJerk SetGearMaster SetGearRatio	SetPositionLoop SetMotorCommand	SetCurrentLoop SetFOCLoop SetCurrentControlMode

Tabelle 18: Doppelt gebufferte Kommandos

* Anmerkung: Das Kommando SetStartVelocity ist nicht doppelt-gebuffert.

-o-

Breakpoints

Breakpoints sind eine günstige Methode, Ereignisse auf einige spezifische Zustände hin zu programmieren. Je nach den Argumenten der Breakpointanweisung kann ein Breakpoint ein Update auslösen; ein *abrupter Stopp* nach einem Update, ein *SmoothStop* nach einem Update, ein Motor OFF gefolgt von einem Update (mehr zu dieser Funktion in einem späteren Abschnitt); oder keine Aktion was auch immer.

Jede Achse kann für die Verwendung von 2 Breakpoints programmiert werden. So können zwei vollkommen gesonderte Zustände überwacht und daraufhin auf Ereignisse ausgelöst werden. Diese zwei Breakpoints werden Breakpoint 1 und Breakpoint 2 genannt. Zusätzlich können jene Register welche in die aktiven kopiert werden sollen unabhängig von einander für den einen oder anderen Breakpoint spezifiziert werden. Das Kommando hierfür lautet SetBreakpointUpdateMask. Zum Lesen der eingestellten Werte verwendet man das Kommando GetBreakpointUpdateMask.

-o-

Einen Breakpoint definieren

Jeder Breakpoint hat fünf Komponenten: die Breakpointachse, die Sourceachse für das Triggerereignis, das Ereignis selbst, die auszuführende Aktion und den Vergleichswert. Diese Komponenten sind in der folgenden Tabelle näher beschrieben.

Breakpoint Komponent	Beschreibung
Breakpoint axis	Ist die Achse, auf der die angegebene Aktion ausgeführt werden soll.

Breakpoint Komponent	Beschreibung
Source axis	Ist die Achse, auf der das Triggerereignis stattfindet. Es kann die Gleiche oder eine andere als die Breakpointachse sein. Irgendeine Anzahl von Breakpoints kann die gleiche Achse wie die Sourceachse benutzen.
Trigger	Ist das Ereignis, der den Breakpoint auslöst.
Action	ist der Aufgaben-Ablauf der von der POSYS® ausgeführt wird, wenn der Breakpoint getriggert wird. Nachdem ein Breakpoint getriggert wird, wird die Aktion auf der Breakpointachse ausgeführt.
Breakpoint update mask	Entweder Profil und/oder Position Loop und/oder Current Loop Werte werden aktualisiert aufgrund eines eventuell auftretenden Breakpointtrigger Ereignisses.
Comparison value	Der Vergleichswert wird benutzt, zusammen mit der Aktion, um das Breakpointereignis zu definieren.

Tabelle 19: Einen Breakpoint definieren

Allzusammen erlauben diese Parameter große Flexibilität um Breakpointzustände zu definieren. Durch das Verknüpfen dieser Komponenten, kann fast jedes Ereignis auf jeder Achse einen Breakpoint verursachen.

Der Befehl SetBreakPoint wird benutzt um die Breakpointachse, den Trigger, die Sourceachse und die Aktion zu definieren. Um diese Werte zurück zu lesen wird der Befehl GetBreakpoint benutzt.

Nach Erhalt des SetBreakpoint Kommandos, wird der Breakpoint aktiv. Das heißt, der Motion Controller wird beginnen, die Bedingungen für den Breakpoint mit den tatsächlichen Gegebenheiten zu vergleichen. Dies bedeutet, dass alle anderen erforderlichen Informationen für die Breakpoint-Funktion (wie Vergleichswert und Breakpoint Update Mask) sollten bereits geladen sein, bevor dieses Kommando ausgeführt wird. Siehe Abschnitt "Breakpoint Beispiele: " Beispiele für die Programmierung von Breakpoints.

Der SetBreakpointMask und SetBreakpointValue-Kommando sollten immer vor dem SetBreakpoint -Kommando ausgeführt werden um einen Breakpoint einzurichten.

-o-

Breakpoint Trigger

Die POSYS 1800-B unterstützt die folgenden Breakpoint Trigger Zustände:

Trigger Kondition	Level oder Threshold	Beschreibung
größer oder gleich positive Sollposition	threshold	Ist gegeben wenn die gegenwärtige Sollposition gleich ist oder größer als der programmierte Vergleichswert.
kleiner oder gleich positive Sollposition	threshold	Ist gegeben wenn die Sollposition gleich ist oder kleiner als der programmierte Vergleichswert.
größer oder gleich aktueller positiver Position	threshold	Ist gegeben wenn die Istposition gleich ist oder größer als der programmierte Vergleichswert.
kleiner oder gleich aktueller negativer position	threshold	Ist gegeben wenn die Istposition gleich ist oder kleienr als der programmierte Vergleichswert.
Sollposition überschritten	threshold	Ist gegeben wenn die Istposition sich kreuzt (ist gleich zu) mit dem programmierten Vergleichswert.
Aktuelle Position überschritten	threshold	Ist gegeben wenn die Istposition sich kreuzt (ist gleich) der programmierte Vergleichswert.
Zeit	threshold	Ist gegeben wenn die gegenwärtige POSYS®-Zeit (in Zyklenzahl seit Einschalten) ist gleich dem programmierten Vergleichswert.
Event status	level	Ist gegeben wenn der EventStatusRegister der Bitmaske und dem high/low Pattern im programmierten Vergleichswert gleicht.
Activity status	level	Ist gegeben wenn der ActivityStatusRegister der Bitmaske und dem high/low Pattern im programmierten Vergleichswert gleicht.
Signal status	level	Ist gegeben wenn der SignalStatusRegister der Bitmaske und dem high/low Pattern im programmierten Vergleichswert gleicht.

Trigger Kondition	Level oder Threshold	Beschreibung
None	%	Deaktiviert einen früher gesetzten Breakpoint.

Tabelle 20: Breakpoint Triggers

Wenn "Keine" für den Breakpointtrigger selektiert wird, bedeutet dies dann tatsächlich, dass jener Breakpoint inaktiv ist. Nur einer der obigen Trigger kann zur gleichen Zeit selektiert werden.

-o-

Threshold-Triggered Beakpoints

Threshold-triggered Breakpoints verwenden den mit dem Kommando SetBreakpointValue ermittelten 32-bit Schwellwert als Vergleichswert. Wenn der Vergleich wahr ist, wird der Breakpoint getriggert. Zum Beispiel wenn es gewünscht wird dass getriggert wird, wenn die Sollposition gleich oder größer 1000000 ist, dann ist der Vergleichswert der mit dem Kommando SetComparisonValue geladen wird 1000000 und der ausgewählte Trigger wäre GreaterorEqualCommandedPosition.

-o-

Level-Triggered Breakpoints

Der Host liefert zwei 16-Bit Daten-Wörter einen level-getriggerten Breakpoint zu setzen: eine Triggermaske und eine Zustandsmaske (high oder low). Diese Werte werden gesetzt mit dem Kommando SetBreakpointValue. Die oberen Bits des Datenwortes sind der Wert für die Triggermaske und die unteren Bits des Datenwortes sind der Wert für die Zustandsmaske.

Die Triggermaske bestimmt, welche Bits des ausgewählten Zustandsregisters für den Breakpoint freigegeben werden.
Eine ''1'' in jeder beliebigen Position der Triggermaske gibt das entsprechende Zustandsregisterbit frei einen Breakpoint zu triggern, eine ''0'' in der Triggermaske sperrt das entsprechende Zustandsregisterbit. Wenn mehr als ein Bit selektiert wird, wird der Breakpoint getriggert dessen spezifizierter Status erreicht wird.

Die Zustandsmaske bestimmt, welcher Status des entsprechenden Statusbits einen Breakpoint erzeugt. Jeder Statusbit, das in dem gleichen Zustand (z.B. 1 oder 0) wie das entsprechende Zustandsbit ist, ist geeignet, einen Breakpoint auszulösen (vorausgesetzt, dass es von der Triggermaske selektiert worden ist).

Zum Beispiel wenn der Statusregisterbreakpoint selektiert worden ist, und die Triggermaske den Wert 0402h und die Zustandsmaske den Wert 0002h enthält wird der Breakpoint dann getriggert wenn Bit 1 (der 'at max. velocity' Indikator) den Wert 1 annimmt, oder Bit 10 (der 'in Bewegung' Indikator) den Wert übernimmt.

-o-

Breakpoint Aktionen

Sobald ein Breakpoint getriggert worden ist, kann die POSYS® programmiert werden, eine der Folgeanweisungen auszuführen:

Aktion	Beschreibung
None	Keine Kommandos werden ausgeführt, jedoch das Breakpoint Bit wird im Event Status Register gesetzt.
Update	Kopiert die doppelt gebufferten Register in die aktiven Register wie vorher mit dem SetBreakpointUpdateMask Kommando gesetzt.
Smooth Stopp	Löst einen Stopp mit Rampe aus entsprechend dem mit den Kommandos SetAcceleration und/oder SetDeceleration gesetzten Werten aus. Der Geschwindigkeitswert wird auf Null (0) gesetzt nach dem der Breakpoint ausgeführt wurde.
Abrupter Stopp	Löst einen abrupten Stopp ohne Rampe aus. Der

Aktion	Beschreibung
	Geschwindigkeitswert wird auf Null (0) gesetzt nach dem der Breakpoint ausgeführt wurde.
Abrupter Stopp mit Löschung des Wertes für den Schleppabstandsfehler	Löst einen abrupten Stopp ohne Rampe aus als auch Löschung des Wertes für einen eventuell aufgelaufenen Schleppabstandsfehler (entspricht dem Kommando ClearPositionError). Der Geschwindigkeitswert wird auf Null (0) gesetzt nach dem der Breakpoint ausgeführt wurde.
Deaktiviere Position Loop & Higher Modules	Deaktiviert den Bahngenerator und Position Loop Module.
Deaktiviere Current Loop & Higher Modules	Deaktiviert den Bahngenerator, Position Loop und Regelschleifenmodule.
Deaktiviere Motorsignalausgabe & Higher Modules	Deaktiviert den Gahngenerator, Position Loop, Regelschleifen- und Motorausgabemodule.

Tabelle 21: Breakpoint Aktionen

Die SetBreakpointUpdateMask bestimmt welche der doppelt-gebufferten Register aktualisiert werden beim Eintreten eines Breakpointereignisses. Der Wert für SetBreakpointUpdateMask wird durch die Auslösung des Breakpointereignisses nicht geändert, jedoch sollte einmal gesetzt und bei dem Wert belassen werden, oder kann auch geändert werden werden nach belieben bzw. entsprechend den Anforderungen der Appplikation angepasst werden.

Für die POSYS 1800-B ist der standardmäßig eingestellte Wert für SetBreakpointUpdateMask den Bahngenerator und die Position Loop Parameter zu aktualisieren.

Für spezifizierte Aktionen die den OperatingMode ändern, muss erneut RestoreOperatingMode ausgeführt werden um wieder in den normalen OperatingMode herzustellen.

Sobald ein Breakpointbedingung erfüllt wurde, wird das entspechende Bit im EventStatusRegister gesetzt und der Breakpoint selbst deaktiviert.

-o-

Breakpoint Latenzen

Die Latenzzeit nachdem ein Breakpoint ausgelöst wurde, hängt von der Bedingung ab die ausgewählt wurde. Breakpoints, die abhängig von internen Motion Control Register sind, haben eine Latenzzeit entsprechend einer Servoloopupdaterate. Breakpoints, die abhängig sind von externen Motion Controller Signalen, haben eine Latenzzeit entsprechend einer doppelten Servoloopupdaterate.

-o-

Breakpoint Beispiele

Hier sind ein paar Beispiele die veranschaulichen sollen wie Breakpoints benutzt werden können.

Beispiel #1: Der Host möchte, dass Achse 1 die Geschwindigkeit verändert, wenn die Enkoderposition einen besonderen Wert erreicht. Breakpoint #1 soll benutzt werden.

Die folgende Kommandosequenz erreicht dies:

This is accomplished using the following command sequence. Beachten Sie, dass diese Sequenz davon ausgeht, dass der UpdateMask auf seinem Standardwert für Bahngenerierung und Position Loop Update belassen wurde.

SetPosition Axis1, 123456	Lade Endposition
SetVelocity Axis1, 55555	Lade Geschwindigkeit
SetAcceleration Axis1, 500	Lade Beschleunigung
SetDeceleration Axis1, 1000	Lade negative Beschleunigung
Update Axis1	Starte Bewegungssequenz. Profil und Position Loop Parameter werden aktualisiert, von denen aber nur die Profilparameter

	geändert wurden.
SetVelocity Axis1, 111111	Lade neue Geschwindigkeit von 111.111 aber noch nicht das Update schicken
SetBreakpointValue Axis1, 0, 100000	Lade 100000 in das Vergleichsregister für Breakpointregister 1
SetBreakPointUpdateMask Axis 1, 0, Profile	Update nur der Profilparameter
SetBreakpoint Axis1, 0, Axis1, Update, GreaterOrEqualActualPosition	spezifiziere einen positiven. aktuellen Positionsbreakpoint auf Achse 1 welches in einem Update endet wenn Breakpoint1 eintritt. Beachten Sie, dass dies das letzte Kommando in der Breakpointdefinitionssequenz ist, da dies der Befehl ist, der den Breakpointvergleich einleiten wird.

Tabelle 22: Breakpoint Beispiel #1

Diese Sequenz startet eine Bewegung und lädt einen Breakpoint, nachdem die erste Bewegung gestartet wurde. Der Breakpoint, der definiert wurde, wird die Geschwindigkeit updaten zu 111111, wenn die Istposition einen Wert von 100000 erreicht. Deswegen wird sich die Achse an Position 100000 von einer Geschwindigkeit von 55555 bis 111111 mit dem Beschleunigungswert von 500 beschleunigen.

Anmerkung: zwischengespeicherte Register, die nicht nochmals gesandt werden, verbleiben in den zwischengespeicherten Registern. Zum Beispiel wenn der Breakpoint ein Update ausführt, sind die Werte für Position, Beschleunigung und Verlangsamung unverändert und werden deswegen zu den aktiven Registern ohne Modifikation kopiert.

Beispiel #2: Der Host möchte, dass Achse 1 einen Notstopp ausführt, wann auch immer das Axisln-Signal für Achse 3 auf „high“ geht. Außerdem die Beschleunigung der Achse 1 soll verändert werden, wann immer eine besondere Istposition auf Achse 4 erreicht wird.

Die folgende Kommandosequenz erreicht dies:

SetPosition Axis1, 123456	Lade Endposition
SetVelocity Axis1, 55555	Lade Geschwindigkeit
SetAcceleration Axis1, 500	Lade Beschleunigung
SetDeceleration Axis1, 1000	Lade negative Beschleunigung
Update Axis1	Starte Bewegung. Profil und Position Loop Parameter werden aktualisiert, von denen aber nur die Profilparameter geändert wurden.
SetBreakpointValue Axis1, 0, 0x400040	Lade Maske und Zustandswort von 0x40, 0x40 (Bit 6 muss “high” sein) für Breakpoint1
SetBreakPointUpdateMask Axis 1, 0, Profile	Update nur der Profilparameter
SetBreakpoint Axis1, 0, Axis3, AbruptStop, SignalStatus	Lade Maske und Zustandswort von 0x40, 0x40 (Bit 6 muss “high” sein) für Breakpoint1
SetAcceleration Axis1, 111111	Lade neue Geschwindigkeit von 111111, aber führe noch kein Update aus
SetPositionLoop Axis1, Kd, 1250	Lade neuen Wert für Kd
SetBreakpointValue Axis1, 1, 100000	Lade den Wert 100000 in das Vergleichsregister für Breakpoint2
SetBreakPointUpdateMask Axis1, 1, Profile 1 Position Loop	Aktualisiere Profilparameter & Position Loop für Breakpoint2
SetBreakpoint Axis1, 1, Axis4, Update, PositiveCommandedPosition	Spezifiziere einen Breakpoint für die positive Sollposition auf Achse 4 welches in einem Update resultiert wenn Zustand gegeben für Breakpoint2

Tabelle 23: Breakpoint Beispiel #2

Diese Sequenz ist dem vorherigen ähnlich, ausser dass ein zusätzlicher Breakpoint definiert worden ist, der den abrupten Stopp veranlasst. Zusätzlich wurden diese Breakpoints definiert um auf Ereignisse zu reagieren die mit den Achsen 3 und 4 zusammenhängen. Beide Breakpoints wurden definiert, nachdem die Bewegung begonnen wurde, obwohl dies nicht unbedingt notwendig ist, je danach, wann man erwartet, dass der Breakpoint eintritt. Generell sollten Breakpoints nach der primären Bewegungsinitiierung eingerichtet werden, weil es nur einen Satz gebufferter Register gibt und es damit unmöglich ist primäre Bewegungsparameter als auch (Position, Geschwindigkeit, etc.) und auch Profilparameter (Parameter die effektiv werden, sobald ein Breakpoint ausgelöst wurde) bevor die eigentliche Bewegung aktualisiert wurde.

-o-

Status Registers

Überblick

Die POSYS® kann fast jeden Aspekt einer Achse überwachen. Es gibt zahlreiche numerische Register, die abgefragt werden können, um die Aktualität der Achse wie die Ist-Position (GetActualPosition-Kommando), die Soll-Position (GetCommandedPosition-Kommando) etc. zu bestimmen. Außer diesen numerischen Registern gibt es drei bit-orientierte Zustandsregister, die einen kontinuierlichen Bericht über den Status einer spezifischen Achse zurückmelden (GetActivityStatus, GetSignalStatus, GetEventStatus).

Der Host kann diese drei Register abfragen, oder die Inhalte dieser Register können in Breakpointoperationen benutzt werden, um ein Triggerereignis wie "trigger wenn Bit 8 im Zustandsregister auf "low" geht“ zu definieren. Diese Register liefern auch die Quelldaten für den AxisOut-Mechanismus, der einem Bit innerhalb dieser drei Register ermöglicht, als ein Hardwaresignal ausgegeben zu werden.

Der Host kann diese 3 Register abfragen, oder der Inhalt dieser Register kann verwendet werden Breakpointaktionen zu definieren, wie z.B. „Trigger Bit 8 sobald es im Signal Status Register von „high“ auf „low“ geht. Diese Register dienen auch als Quelle für den AxisOut-Mechanismus (siehe „SetEventAction“), welches erlaubt eine oder mehrere dieser Bits innerhalb dieser drei Register als Hardwaresignal ausgegeben zu werden.

-o-

Event Status Register

Das Ereignis-Zustandsregister (Event Status Register) ist konstruiert, um Ereignisse zu registrieren, die sich nicht kontinuierlich ändern, aber dazu neigen bei einem spezifischen Ereignis vorzukommen. Als solcher, wird jedes der Bits im Register von der POSYS® gesetzt und vom Host gelöscht.

Das Ereignis-Zustandsregister wird in der Tabelle unten beschrieben:

Bit	Name	Beschreibung
0	Motion complete	Wird gesetzt wenn ein Profil abgeschlossen wurde. Die Bewegung, die für vollendet erachtet wird, kann auf der Soll-Position oder Ist-Position basieren.
1	Position wraparound	Wird gesetzt wenn die Ist-Position 7FFFFFFFh überschritten (die positivste Stellung) wird und konvertiert zu 80000000h (die negativste Stellung), oder umgekehrt.
2	Breakpoint 1	Wird gesetzt wenn Breakpoint #1 getriggert wird
3	Capture received	Wird gesetzt, wenn der high-speed Latch Input eine neue Positionsangabe erhält.
4	Motion error	Wird gesetzt wenn die Ist-Position von der Soll-Position mehr differiert als der Schleppabstandswert erlaubt.
5	Positive limit	Wird gesetzt wenn der positive Endschalter angefahren wird.
6	Negative limit	Wird gesetzt wenn der negative Endschalter angefahren wird.
7	Instruction error	Wird gesetzt wenn ein Kommandofehler auftritt
8-12	n/a	Nicht verfügbar POSYS 1800-B
13	Reserviert	Kann 1 oder 0 sein
14	Breakpoint 2	Wird gesetzt wenn Breakpoint #2 getriggert wird
15	Reserviert	Kann 1 oder 0 sein

Tabelle 24: Event Status Register Bits

Das Kommando GetEventStatus liefert die Inhalte des Ereignis-Zustandsregisters für die angegebene Achse.

Bits im Ereignis-Zustandsregister werden gelatcht. Sobald gesetzt, bleiben sie gesetzt bis sie vom einem Hostkommando oder einem Systemreset gelöscht werden. Ereignis-Zustandsregisterbits können vom Kommando ResetEventStatus auf 0 zurückgesetzt werden, indem eine 16-Bit Maske benutzt wird. Register-Bits, die 0 in der Maske entsprechen, werden resetted; alle anderen Bits bleiben unbeeinflusst.

Das Ereigniszustandsregister kann auch benutzt werden, einen Interrupt zu generieren. Das Kommando SetInterruptMask kann hierfür verwendet werden.

-o-

Instruktionsfehler

Bit 7 des Ereigniszustandsregisters zeigt einen Anweisungsfehler an. Solch ein Fehler kommt vor, wenn eine sonst zulässige Anweisung oder Anweisungsfolge gesandt wird, wenn der gegenwärtige operative Zustand der POSYS® die Anweisung ungültig macht. Anweisungsfehler kommen immer nur zum Zeitpunt eines Updates vor. Sollte ein Anweisungsfehler auftreten, werden die ungültigen Parameter ignoriert, und der Anweisungs-Fehler-Indikator (Instruction Error Indicator) des Ereignis-Zustandsregisters wird gesetzt. Während ungültige Parameter während des Updates überprüft und ignoriert werden, werden zulässige Parameter weiterhin geschickt und akzeptiert. Dies kann unabsichtliche Nebenwirkungen je nach Art der Bewegung nach sich ziehen, sodass alle Anweisungsfehlerereignisse sehr ernst genommen werden sollten.

Im folgenden Beispiel wird die negative Geschwindigkeitsvorgabe im neuen Profil nicht akzeptiert.

SetProfileMode (axis2, Velocity)	Setze Profile Mode auf Geschwindigkeitsmodus
SetVelocity (axis2, –4387)	Setze Geschwindigkeit auf einen negativen Wert
SetUpdateMask(axis2, Trajectory)	Setze die UpdateMask
Update (axis2)	Führe Update aus
SetProfileMode (axis2, Trapezoidal)	Wechsele den Profilmodus zu Trapez
SetPosition (axis2, 123456)	Lade neue Position
Update (axis2)	Führe Update aus

Tabelle 25: Instruktionsfehler

Der Update wird ausgeführt, aber das Instruction Error Bit wird gesetzt. Legitime Parameter wie die neue Position werden aktualisiert und der Bahngenerator arbeitet weiter.

-o-

Activity Status Register

Wie das Ereignis-Zustandsregister überwacht das Aktivitäts-Zustandsregister verschiedene Prozesse der POSYS 1800-B.

Aktivitäts-Zustandsregisterbits werden allerdings nicht gelatcht, sie werden kontinuierlich von der POSYS® gesetzt und zurückgesetzt um den gegenwärtigen Status anzuzeigen.

Das Activity Status Register wird in der Tabelle unten beschrieben:

<table>
<tr><th>Bit</th><th>Name</th><th>Beschreibung</th></tr>
<tr><td>0</td><td>Phasing initialized</td><td>Gesetzt (1) sobald die Hardware der Motorkommutierung initialisiert wurde. Gelöscht (0) falls noch nicht initialisiert.</td></tr>
<tr><td>1</td><td>At maximum velocity</td><td>Gesetzt (1) wenn die Sollgeschwindigkeit gleich der Istgeschwindigkeit ist, wie vom Host angegeben.
Gelöscht (0), wenn es nicht der Fall ist. Dieses Bit funktioniert nur im Trapez-, Geschwindigkeits-, oder S-Kurvenmodus. Es funktioniert nicht im Electronic-Gearmodus</td></tr>
<tr><td>2</td><td>Position tracking</td><td>Gesetzt (1) wenn der Servo die Achse innerhalb des Tracking Windows behält. Gelöscht (0), wenn es nicht der Fall ist.</td></tr>
<tr><td>3-5</td><td>Current profile mode</td><td>Diese Bits zeigen den gegenwärtigen Profilmodus an, der anders sein könnte als der Wert, der benutzt wurde mit dem Kommando SetProfileMode, nach dem ein Update-Befehl noch nicht erfolgte.
<table>
<tr><th>Bit 5</th><th>Bit 4</th><th>Bit 3</th><th>Profile Mode</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>trapezoidal</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>velocity contouring</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>S-curve</td></tr>
</table>
</td></tr>
</table>

Bit	Name	Beschreibung			
		0	1	1	electronic gear
6	Reserved	Darf 1 oder 0 sein			
7	Axis settled	Gesetzt (1) wenn sich die Achse innerhalb des SettleWindows für eine vorher spezifizierte Zeitspanne aufhielt. Gelöscht (0) wenn dies nicht der Fall ist.			
8	Position Loop Enabled	Gesetzt (1) wenn entweder das Position Loop Module oder der Bahngenerator aktiviert sind. Cleared (0) wenn beides deaktiviert ist. Das Kommando SetOperatingMode wird üblicherweise verwendet welche aktiviert oder deaktiviert sind; jedoch können Module automatisch deaktiviert werden durch Event Actions (SetEventAction Kommando) oder Breakpoints.			
9	Position capture	Gesetzt (1) wenn ein neuer Positionswert verfügbar ist, der von der high-speed Latch Input Hardware gelesen werden kann. Gelöscht (0) wenn ein neuer Wert noch nicht erfasst wurde. Während dieses Bit gesetzt ist, werden keine neuen Werte erfasst. Der Befehl GetCaptureValue erfasst den Wert und löscht das Bit. Damit ist die nächste Erfassung möglich.			
10	In-motion indicator	Gesetzt (1) wenn die Soll-Position des Trajektorieprofils sich ändert. Gelöscht (0) wenn die Soll-Position sich nicht ändert. Der Wert dieses Bits kann oder darf nicht dem Wert des Motion Complete Bits des Ereigniszustandsregisters entsprechen, unabhängig davon ob der Motion Complete Mode auf Soll- oder Ist-Wert gesetzt wurde.			
11	In positive limit	Gesetzt (1) wenn der Motor den positiven Endschalter erreicht hat. Gelöscht (0) wenn dem nicht so ist.			
12	In negative limit	Gesetzt (1) wenn der Motor den negativen Endschalter erreicht hat. Gelöscht (0) wenn dem nicht so ist.			
13-15	Profile segment	Zeigt die Nummer der einzelnen Segmente der S-Kurve mit den Werten 1-7 an, wie gezeigt in der S-Kurvenbeschreibung dieses Handbuches. Ein Wert von 0 in diesem Feld zeigt, dass die Bahn nicht in Bewegung ist. Dieses Feld ist undefiniert für andere Profilmodi als der S-Kurve und kann 0's oder 1's enthalten.			

Tabelle 26: Activity Status Register

Das Kommando GetActivityStatus liefert die Werte des Aktivitäts-Zustandsregisters für die angegebene Achse.

-o-

Signal Status Register

Das Signal Status Register bietet Real-Time Signalerfassung für verschiedene Eingänge. Das Signal Status Register wird in der Tabelle unten beschrieben:

Bit	Name	Beschreibung
0	A encoder	A-Signal des Enkodereingangssignals
1	B encoder	B-Signal des Enkodereingangssignals
2	Index encoder	Index-Signal des Enkodereingangssignals
3	Home/Capture	Referenzschaltereingang
4	Positive limit	Positiver Endschaltereingang
5	Negative limit	Negativer Endschaltereingang
6	AxisIn	Frei-programmierbarer achsenspezifischer Eingang
7	Hall1	Hall Sensor Eingang 1.
8	Hall2	Hall Sensor Eingang 2.
9	Hall3	Hall Sensor Eingang 3.
10	AxisOut	Frei-programmierbarer achsenspezifischer Ausgang
11-12	Reserved	Darf 0 oder 1 sein.

Bit	Name	Beschreibung
13-14	n/a	Nicht für POSYS 1800-B verwendbar.
15	Reserved	Darf 0 oder 1 sein.

Tabelle 27: Signal Status Register

Das Kommando GetSignalStatus liefert den Inhalt des Signal Status Registers der spezifizierten Achse. Alle Signal Status Register Bits sind Eingänge mit Ausnahme von Bit 10 (AxisOut).

Die Bits in den Signal Status Register stellen immer das eigentliche Hardwareniveau auf den entsprechenden Eingängen dar. Eine ''1'' in diesem Register stellt einen elektrisch hohen Wert auf dem Eingang dar; eine ''0'' zeigt ein elektrisch niedriges Niveau an. Der Zustand der Signal Sense Mask beeinflusst den Wert der gelesen wurde mit dem Kommando GetSignalStatus (lesen Sie bitte im nächsten Abschnitt mehr über die Signal Sense Mask).

Die eigentliche Interpretation des Signals ist abhängig von seiner Funktion. Zum Beispiel sind Index, Home, Negative Limit und Positive Limit als aktiv low ausgelegt, d.h. wenn eine 0 gelesen wird mit dem Kommando GetSignalStatus bedeutet dies dass das Signal aktiv ist. Wenn man das Beispiel mit dem Signal für Negative Limit anwendet, wenn GetSignalStatus einen Wert von 0 aufweist, interpretiert der Motion Controller dies so, dass die entsprechende Achse den negativen Endschalter aktiviert hat. Andere Signale wie HallA, HallB, HallC, AxisIn und AxisOut verfügen nicht über ein Active „high“ oder active „low“ Status als solches. In den entsprechenden Abschnitten des Handbuchs erfahren Sie mehr wie diese Signale vom Motion Controller verwendet werden.

-o-

Signal Sense Mask

Die Bits in dem Signal Status Register stellen den high/low Zustand von verschiedenen E/As auf der POSYS® dar. Wie diese E/As von der POSYS® interpretiert werden, kann durch die Verwendung der Signal Sense Mask definiert werden. Dies ist nützlich um die Interpretation von Eingangsignalen zu verändern um sich wiederum der Interpretation der Signale der Hardware des Anwenders anzupassen. Der Standardwert für SignalSenseMask ist „nicht invertiert“ ausser für das Index-Signal dass standardmäßig „invertiert“ ist.

Das SignalSenseMask-Register wird in der Tabelle unten beschrieben:

Bit	Name	Beschreibung
0	A encoder	Gesetzt (1) um A-Enkodereingangssignal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
1	B encoder	Gesetzt (1) um B-Enkodereingangssignal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
2	Index encoder	Gesetzt (1) um Indexenkodersignal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
3	Home/Capture	Gesetzt (1) um Referenzsignal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
4	Positive limit	Gesetzt (1) für eine "active high" Interpretation des positiven Endschalters, welches bedeutet, dass der Endschalter für die positive Fahrtrichtung angefahren wurde, wenn das Signal auf "active high" geht. Kein Endschalter (0) wenn "active low".
5	Negative limit	Gesetzt (1) für eine "active high" Interpretation des negativen Endschalters, welches bedeutet, dass der Endschalter für die negative Fahrtrichtung angefahren wurde, wenn das Signal auf "active high" geht. Kein Endschalter (0) angefahren wenn "active low".
6	AxisIn	Gesetzt (1) um das AxisIn Signal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
7	Hall1	Gesetzt (1) um das HallA Signal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
8	Hall2	Gesetzt (1) um das HallB Signal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
9	Hall3	Gesetzt (1) um das HallC Signal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
10	AxisOut	Gesetzt (1) um das AxisOut Signal zu invertieren. Nicht gesetzt (0) für keine Invertierung.
11	Step Output	Gesetzt (1) um die aktive Änderung als low-to-high zu definieren. Nicht gesetzt (0) um die aktive Änderung als high-to-low zu definieren.
12	Motor Direction	Gesetzt (1) um die Motorrichtung zu invertieren.

Bit	Name	Beschreibung
		Nicht gesetzt (0) für keine Invertierung.
13-15	Reserved	

Tabelle 28: Signal Sense Mask

Falls mit dem Kommando SetCaptureSource das Index-Signal als Capture Source gesetzt wurde um das Index-Signal zu erfassenis, sollte Bit 2 (Index Signal Sense) verwendet werden, um die Polarität des Index zu steuern.

-o-

Bewegungsüberwachung und verwandte Aufgaben

SetEventAction Prozess

Die POSYS® 1800/1900 unterstützen einen programmierbaren Mechanismus um auf sicherheits- oder leistungsbezogene Bedingungen zu reagieren.

Das Kommando SetEventAction definiert die auszuführende Aktion bei einer bestimmten Bedingung. Um eine ereignis-bedingte Antwort zu definieren, müssen sowohl die Bedingung als auch die Aktion spezifiziert werden. Die folgende Tabelle listet die möglichen Event Status Register Bedingungen auf die verwendet werden können um eine ereignis-bedingte Aktion zu definieren.

Bedingung	Beschreibung
Motion error	Ein Bewegungsfehler tritt ein wenn ein programmierter Schleppabstandsfehler überschritten wird.
Positive limit	Ein positives Endschaltereregnis tritt ein sobald das korrespondierende Signal aktiv wird solange der Motor eine positive Geschwindigkeit aufweist.
Negative limit	Ein negatives Endschaltereregnis tritt ein sobald das korrespondierende Signal aktiv wird solange der Motor eine negative Geschwindigkeit aufweist.

Tabelle 29: SetEventAction Prozess

Beides, Motion Error Processing und Endschaltereregnisse sind in den entsprecenden Kapiteln näher beschrieben. Zusätzlich zu diesen vier überwachten Bedingungen, ist es möglich eine sofortige sicherheits-bedingte Aktion auszuführen. Siehe POSYS 1800-B Programmier Handbuch für weitergehende Informationen.

Action Name	Beschreibung
No Action	Keine Aktion wird ausgeführt.
Smooth Stop	Veranlasst einen Stopp mit programmierte Rampe. Die Geschwindigkeit wird auf Null (0) gesetzt, nachdem die EventAction augeführt wurde.
Abrupt Stop	Veranlasst einen sofortigen Stopp des Trajektoriegenerators. Die Geschwindigkeit wird auf Null (0) gesetzt, nachdem die EventAction ausgeführt wurde.
Abrupt Stop with Position Error Clear	Veranlasst einen sofortigen Stopp des Trajektoriegenerators als auch die Löschung des Schleppabstandsfehlers (= ClearPositionError). Die Geschwindigkeit wird auf Null (0) gesetzt, nachdem die EventAction augeführt wurde.
Disable Position Loop & Higher Modules	Deaktiviert den Trajektoriegenerator und Position Loop Module.
Disable Current Loop & Higher Modules	Deaktiviert den Trajektoriegenerator, Position Loop und Current Loop Module .
Disable Motor Output & Higher Modules	Deaktiviert den Trajektoriegenerator, Position Loop, Current Loop und Motorausgabemodule.

Tabelle 30: SetEventAction Aktion

Nachdem der Eventaction programmiert wurde, wird der Motion Prozessor die spezifizierten Konditionen permanent überwachen und die programmierte Aktion ausführen, sobald das Ereignis eintritt. Des weiteren werden verwandte Aktionen ausgeführt, wie z.B. das Setzen des entsprechenden Bits im Event Status Register.

Um eine EventAction aufzuheben wird das Kommando RestoreOperatingMode verwendet. Dieses Kommando setzt den Motion Prozessor in den Operating Mode zurück der vorher mit dem Kommando SetOperatingMode definiert wurde. Bitte beachten Sie, dass falls die EventAction immer noch aktuell ist, wird sie sofort wieder ausgeführt.

Es liegt in der Verantwortung des Anwenders die Bedingung des ereignis-bedingten Events zu eruieren und aufzulösen, ehe wieder mit RestoreOperatingMode der Motor wieder in Betrieb genommen werden kann.

Falls als EventAction No Action, *Abrupt Stop* oder *Smooth Stop* programmiert wurde, wird das Kommando RestoreOperatingMode keinen Einfluss auf die Wiederherstellung des Betriebes haben. Es hat nur Einfluss auf die Wiederherstelluing der Module allein.

Wenn erst programmiert, bleibt ein EventAction aktiv bis es reprogrammiert wurde. Das Auftreten einer Ereignisbedingung setzt die definierte EventAction nicht zurück, sie bleibt aktiv.

Die POSYS® 1800-B Karten stellen Standardwerte für die ereignis-bedingte Verarbeitung zur Verfügung. Diese Standardwerte sind gedacht einen sicheren Betrieb für viele typische Motion Control Anwendungen bereit zu stellen. Ob diese Standardwerte der Anwendung genügen, muss der Anwender bestimmen.

Die Standardwerte lauten wie folgt:

Bedingung	Standardaktion
Motion Error	Deaktiviere Position Loop und Trajektoriegenerator
Positive & Negative Limit	Abrupter Stopp mit Löschung des Schleppabstandfehlerwertes.

Tabelle 31: Standard Event Actions

-o-

Schleppabstandsfehler

Unter gewissen Umständen kann sich die Istposition (Enkoderstellung) von der Sollposition (aktuelle Sollpositionsvorgabe des Trajektoriegenerators) in erheblichem Umfang unterscheiden. Solch ein übermäßiger Positionsfehler zeigt oft einen potenziell gefährlichen Zustand wie Motor- oder Enkoderausfall, oder übermäßige mechanische Reibung an.

Um diesen Zustand zu entdecken, der die persönliche und Anlagensicherheit zu erhöhen helfen soll, verfügt die POSYS® über einen programmierbaren maximalen Schleppabstandsfehler.

Das Kommando SetPositionErrorLimit wird verwendet um den maximalen Schleppabstandsfehler zu programmieren und der programmierte Wert wird mit dem Kommando GetPositionErrorLimit ausgelesen. Um zu bestimmen, ob ein Schleppabstandsfehler vorliegt, wird kontinuierlich die Istposition mit der Sollposition verglichen. Wenn der Schleppabstandsfehlergrenzwert überschritten wird, spricht man von einem "Schleppabstandsfehler".

In dem Augenblick in dem ein Schleppabstandsfehler passiert, werden weitere Aktionen gleichzeitig gestartet. Die folgende Liste beschreibt welche Ereignisse stattfinden:

- Der "Motion Bit Error" des "Event Status Word" wird gesetzt.

- Falls die Standardeinstellung für Event Action bei Motorfehler nicht verändert wurde, werden der Bahngenerator und die Position Loop Module deaktiviert. Bei Achsen im Servomodus bedeutet dies dass die manuell gesetzte Motorvorgabespannung ausgegeben wird (Open Loop Mode). Für Schrittmotoren bedeutet es dass sie zu einem sofortigen Stopp gebracht werden. Falls ein neuer Event Action programmiert wurde, wird dann dieses ausgeführt.

Um einen Schleppabstandsfehler zu lösen, der zum Ergebnis hat, dass der Motor ausgeschaltet wird, sollte der Grund für den Schleppabstandsfehler erkundet werden. Falls das Ereignis dazu führte Kontrollmodule zu deaktivieren, dann sollte der Host mit der Ausführung des Kommando's RestoreOperatingMode den vorhergehenden Zustand wieder herzustellen.

Falls als EventAction No Action programmiert wurde, wird nur der Motion Error Status Bit gesetzt. In diesem Fall wird keine Wiederherstellungssequenz benötigt um den Motion Controller weiter arbeiten zu lassen. Jedoch sollte der Anwender den Grund für das Ereignis ermitteln und die Bewegung manuell beenden.

Nach einem Schleppabstandsfehler wird das entsprechende Bit im EventStatusRegister gelöscht mit der Ausführung des Kommando's ResetEventStatus. Für EventActions wie SmoothStop, AbruptStop und AbruptStop with Position Error Clearing ist es notwendig dieses Bit zurückzusetzen, ehe weitere Bewegungen ausgeführt werden können. Obwohl empfohlen, für EventActions die Module deaktivieren ist es nicht notwendig dieses Bit zurückzusetzen um weitere Bewegungen auszuführen, auch ist es nicht benötigt falls No Action definiert wurde als EventAction.

-o-

Endschalter

Die POSYS® unterstützt Endschalterereignisse die verwendet werden können, um automatisch eine "Bewegungsende"-Bedingung zu erkennen. Dies ist ein wichtiges Sicherheitsmerkmal für Positioniersysteme, die eine definierte Reichweite haben.

Die positiven und negativen Endschalter werden mit POSYS®-Eingängen Poslim1-4 beziehungsweise Neglim1-4 verbunden, um das Anschlagen am positiven bzw. negativen Endschaltereingang zu entdecken.

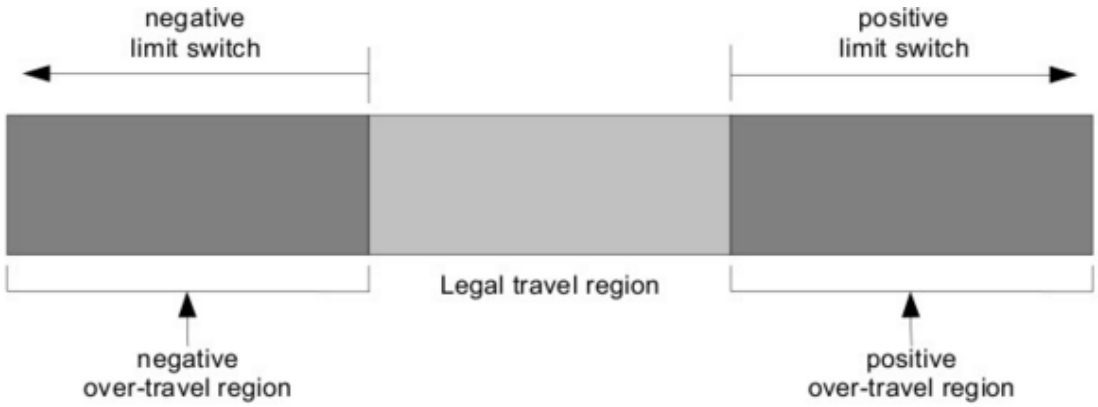

Abbildung 18: Richtungsabhängiger Endschalterüberwachung

Weitere Informationen über die Anbindung dieser Signale finden Sie im Installationshandbuch für die POSYS 1800-B.

Im dem Moment in dem ein positives oder negatives Endschalterereignis auftritt, werden mehrere Ereignisse gleichzeitig ausgelöst. Das entsprechende Endschalter-Bit des Event Status Register wird zusammen mit dem entsprechendem Bit im Activity Status Register gesetzt. Falls für positive und negative Endschalterereignisse EventAction nicht modifiziert wurden, dann wird der Bahngenerator einen abrupten Stopp ausführen und der aufgelaufene Schleppabstandsfehler wird gelöscht. Falls ein neues EventAction programmiert wurde, dann wird dieses ausgeführt.

Falls als EventAction No Action programmiert wurde, wird nur das Endschalterbit gesetzt. In diesem Fall wird keine Wiederherstellungssequenz benötigt um den Motion Controller weiter arbeiten zu lassen. Jedoch sollte der Anwender den Grund für das Ereignis ermitteln und die Bewegung manuell beenden.

Falls die Endschalterüberwachung der POSYS® eingeschaltet wurde, werden die Eingänge kontinuierlich auf ein Endschalterereignis hin überwacht. Ein Endschalterereignis entsteht dann wenn ein Endschaltereingang aktiv wird, wenn sich die Achse in Richtung der Sollposition des Endschalters bewegt. Falls sich die Achse nicht bewegt, in Open-Loop-Mode befindet oder sich in die entgegengesetzte Richtung bewegt, wird kein Endschalterereignis eintreten. Z.B. wird ein Endschalterereignis dann eintreten, wenn die Sollposition sich in positiver Richtung befindet und der Endschalter für die positive Richtung auf "high" schaltet. Jedoch wird dies nicht eintreten wenn die Sollposition sich in negativer Richtung befindet oder die Achse sich nicht bewegt.

Der Zustand der Endschaltereingänge (active high oder active low) kann mit dem Kommando SetSignalSense definiert werden.

Nachdem ein Endschalterereignis eingetreten ist, sollten folgende Schritte unternommen werden, um das Ereignis zu klären:

1. Außer wenn Endschalterereignisse während normaler Maschinenoperation vorkommen können, sollte die Ursache des Ereignisses untersucht werden und entsprechende Sicherheitskorrekturen gemacht werden.

2. Der/die Endschalterbit(s) im "Event Status Register" sollte(n) durch das Kommando ResetEventStatus gelöscht werden. Falls als EventAction No Action programmiert wurde, ist keine weitere Bewegung mehr möglich während ein Endschalterbit im "Event Status Register" gesetzt ist.

3. Falls der standardmäßige EventAction Abrupt Stop with Position Error Clear nicht geändert wurde, sollte eine Bewegung in die entgegengesetzte Richtung initiiert werden, welche das Endschaltereignis ausgelöst hat. Dieses kann jede Profilbewegung sein die vom Endschalter herunterführt. Sollte aber die Achse den Versuch unternehmen weiter in die Richtung auf den Endschalter zu fahren, wird ein neues Endschalterereignis und ein Kommando-Fehler ausgelöst. Siehe Kapitel “Instruktionsfehler” für weiterführende Informationen bzgl. fehlerhafter Anweisungen. Anzumerken gilt noch der Hinweis, dass als Teil des EventActions ein Abrupt Stop assoziiert ist und dadurch das Velocity Register auf Null (0) gesetzt wird. Daher muss das Kommando SetVelocity ausgeführt werden um eine neue Geschwindigkeit zu laden mit der aschliessend vom Endschalter heruntergefahren werden kann.

4. Falls EventAction dahingehend geändert wurde im Vergleich zum Standard, dass ein Modul deaktiviert wird, muss anschliessend das Kommando RestoreOperatingMode ausgeführt werden um normale Operationsbedingungen wieder herzustellen. Dann erst kann eine Bewegung in entgesetzter Richtung ausgeführt werden.

Wenn ein EventAction *No Action* für Endschalter definiert wurde, dann ist nur Schritt 1 erforderlich.

Für Achsen im Electronic Gearing Mode müssen obige Schritte leicht modifiziert werden. Nach einem ResetEventStatus muss noch das Kommando SetGearRatio gefolgt von einem Update-Kommando ausgeführt werden. Dies ist erforderlich weil nach einem Endschaltereereignis das GearRatio auf Null (0) gesetzt wurde.

Falls ein Endschalter sowohl mit dem positiven als auch negativen Eingang der POSYS® verkabelt sind, ist es nicht mehr möglich weder in die eine noch in die andere Richtung herunter zu fahren. Jedoch, falls diese Anordnung gewünscht wird (vermutlich aufgrund einer besonderen Verdrahtungsanordnung), werden dann beide Endschalterbits im Activity Status Register gesetzt, welches zur Folge hat, daß eine Bewegung weder in die eine noch in die andere Richtung möglich ist. In diesem Fall sollte das Kommando SetEventAction *No Acton* oder das Kommando SetLimitSwitchMode verwendet werden um die Endschalterüberwachung zu deaktivieren, während der Motor vom Endschalter heruntergefahren wird.

Im Regelfall werden Endschaltereereignisse verwendet, während das Bahngeneratormodul aktiviert ist. Falls das Bahngeneratormodul deaktiviert ist (z.B. während der Verstärkerkalibrierung), ist zwar die Endschaltererkennung aktiviert, aber bei einer möglichen Erkennung sind EventActions limitiert auf das Deaktivieren anderer Module. Der Bahngenerator muss aktiviert sein für *Smooth Stop*, *Abrupt Stop* oder *Abrupt Stop with Position Error Clear* um automatisch ausgelöst zu werden.

-o-

Tracking Window

Die POSYS® stellt programmierbare Tracking Windows zur Verfügung, die verwendet werden können das Servoverhalten außerhalb des Zusammenhanges eines Schleppabstandsfehlers zu überwachen. Das "Tracking Window" funktioniert ähnlich wie beim Schleppabstandsfehler, dadurch, dass es eine programmierbare Positionierfehlergrenze gibt, innerhalb derer die Achse verweilen muss.

Allerdings, im Gegensatz zum programmierbaren Schleppabstandsfehler, falls die Achse sich außerhalb des "Tracking Windows" bewegt, wird die Achse nicht gestoppt. Das "Tracking Window" ist nützlich, wenn externe Prozesse von dem Motor abhängen, bestimmte Regelparameter innerhalb eines Bereiches zu ermitteln. Alternativ kann das "Tracking Window" als eine Art Frühwarnung für Positionierprobleme benutzt werden, die sich noch nicht als Schleppabstandsfehler qualifizieren lassen.

Die Größe des "Tracking Windows" (maximaler erlaubter Positionierfehler innerhalb des "Tracking Windows") wird mit dem Kommando SetTrackingWindow gesetzt. Der Befehl GetTrackingWindow liefert den programmierten Wert zurück.

Wenn der Positionierfehler kleiner als oder gleich dem Wert für das "Tracking Window" ist, ist der Bit im Activity Status Register gesetzt. Falls der Positionierfehler den programmierten Wert überschreitet, wird das Bit gelöscht.

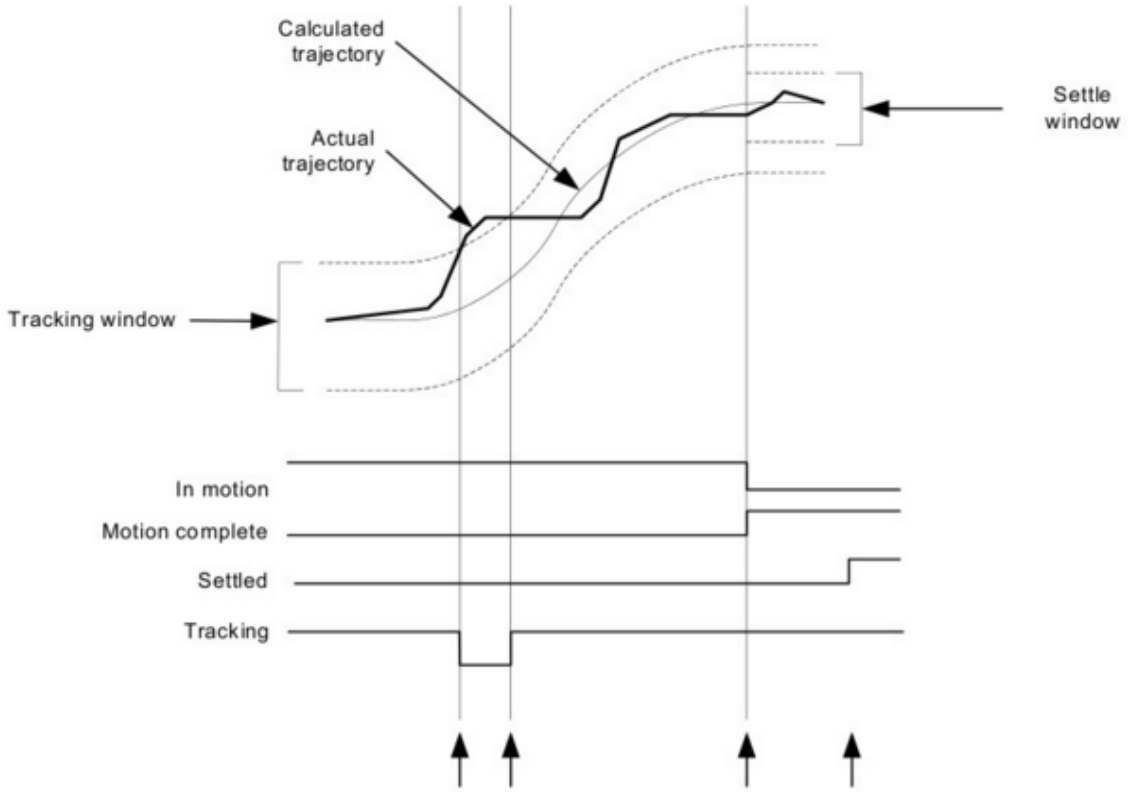

Abbildung 19: Tracking Window

-o-

Motion Complete Indikator

In vielen Fällen ist es nützlich, den Regler signalisieren zu lassen, dass eine vorgegebene Bewegung abgeschlossen wurde. Diese Funktion ist verfügbar als "Motion Complete Indicator".

Der "Motion Complete Indicator" erscheint in Bit 0 des "Event Status Register". Wie alle Bits im "Event Status Register" wird der "Motion Complete Bit" gesetzt durch den Controller und vom Host wieder zurückgesetzt. Wenn eine Bewegung abgeschlossen worden ist, setzt der Controller das "Motion Complete Bit". Der Host kann dieses Bit durch das Abfragen des "Event Status Register" prüfen, oder der Host kann einige automatische Folgeaufgaben programmieren, die einen Breakpoint, einen Interrupt, oder ein AxisOut-Signal benutzen. In jedem Fall sollte der Host, sobald der Host erkannt hat, dass die Bewegung abgeschlossen worden ist, das "Motion Complete Bit" löschen, welches das Ende der Bewegung signalisiert. Diese Aktion erlaubt das Bit das Ende einer Bewegung zu signalisieren für ein neue Bewegung.

"Motion Complete" kann das Ende der Bewegung in einer von zwei Arten anzeigen. Die 1. Möglichkeit ist Kommando-basiert; der "Motion Complete Indicator" wird alleine auf Basis des Trajektoriegenerators gesetzt. Die andere Methode basiert auf die aktuelle Situation, welches bedeutet dass der "Motion Complete Indicator" auf die aktuelle Enkoderposition bezogen wird. Das Kommando SetMotionCompleteMode bestimmt die Art der Bedingung die den Indikator aktiviert.

Wenn es auf den Sollwert basiert, wird die Bewegung für beendet angesehen, wenn die Sollgeschwindigkeit und -beschleunigungswerte beide Null (0) ergeben. Dies geschieht normalerweise am Ende einer Bewegung wenn die Sollposition erreicht worden ist. Aber es kann auch geschehen als das Ergebnis eines Stoppbefehles (SetStopMode -Befehl), einer Geschwindigkeitsänderung zu null, oder wenn ein Endschalter angefahren wird.

Wenn es auf die aktuelle Enkoderposition basiert, wird die Bewegung als beendet betrachtet, wenn alle folgenden Bedingungen erfüllt sind:

- Die Sollposition wurde erreicht.

- Die Differenz zwischen der Ist-Position und der Soll-Position ist weniger als oder gleich dem mit SetSettleWindow definierten Wert. Dieser Wert kann zurückgelesen werden mit dem Kommando GetSettleWindow.

- Die beiden vorhergehenden Bedingungen wurden für eine kontinuierliche Periode von X-Zyklen erfüllt, wobei X der programmierte Wert für "SettleTime" ist. Dieser Wert wird gesetzt/programmiert mit dem Kommando SetSettleTime und kann mit dem Kommando GetSettleTime zurückgelesen werden.

Am Ende des Bewegungsprofils wird der Periodenzeitgeber für den Enkoder-basierten Bewegungsendemechanismus gelöscht. Dadurch wird es mindestens X Periodenverzögerungen (durch SetSettleTime definiert) geben zwischen dem Profilgenerator, der zu einem Ende gekommen ist und dem Zeitpunkt zu dem der "Motion Complete Bit" gesetzt wird. Der "Motion Complete Bit" funktioniert nur im S-Kurven-, Trapez- und Geschwindigkeitsmodus. Es funktioniert nicht im Electronic Gearing Modus.

Beim Enkoder-basierten Motion Complete Mode muss eine angemessene Programmierung gewährleistet sein , weil es durchaus möglich ist, dass der Motion Complete Bit nie gesetzt wird, falls der Servoregelkreis nicht genau genug regelt, um innerhalb des programmierten Positionsfehlerfensters für die spezifizierte Zeit zu verbleiben.

-o-

In-Motion Indikator

Die POSYS® kann feststellen ob die Achse in Bewegung ist oder nicht. Dies wird durch den "In-Motion Indicator" bewerkstelligt.

Der "In-Motion Indicator" erscheint im Bit 10 im "Activity Status Register". Der "In-Motion Bit" ist ähnlich dem "Motion Complete Bit", jedoch gibt es 2 signifikante Unterschiede. Der 1. Unterschied ist (wie bei allen Bits im "Activity Status Register") der "In-Motion Indicator" zeigt kontinuierlich den Status ohne Interaktion mit dem Host an. Mit anderen Worten, der "In-Motion Bit" kann nicht durch den Host gesetzt oder zurückgesetzt werden. Der andere Unterschied ist, dass dieser Bit den Soll-Bewegungszustand des Trajektoriegenerators anzeigt und nicht den aktuellen Enkoderzustand.

Der "Motion Complete Bit" funktioniert nur im S-Kurven-, Trapez- und Geschwindigkeitsmodus. Es funktioniert nicht im Electronic Gearing Modus.

Es ist empehlenswert den Motion Complete Bit zu verwenden um zu bestimmen ob eine Bewegungsequenz zu Ende geführt wurde oder nicht.

-o-

Settle Window

Die POSYS® kann auch kontinuierlich anzeigen ob oder ob nicht die Achse still steht.

Der "Settled Indicator" erscheint im Bit 7 des "Activity Status Register". Der "Settled Indicator" ist ähnlich dem "Motion Complete Bit" wenn dieser im "Actual-Modus" läuft. Die Differenzen sind dass der "Settled Indicator" kontinuerlich seinen Zustand anzeigt (kann nicht gesetzt oder gelöscht werden) und auch anzeigt ungeachtet ob oder ob nicht der "Motion Complete Mode" auf "aktuell" gesetzt wurde.

Die Achse wird für als "In-Position" gehalten, wenn die Achse steht (z.B. eine Bewegungssequenz für die Achse wird nicht ausgeführt) und wenn die aktuelle Position des Motors sich in der Soll-Position für die programmierte "SettleTime" befindet.

"SettleWindow" und "SettleTime" zusammen mit dem "Settled Indicator" sind das selbe wie "Motion Complete Bit". Entsprechend werden die gleichen Kommandos verwendet um die Werte zu setzen oder zurück zu lesen: SetSettleWindow, GetSettleWindow, SetSettleTime, GetSettleTime.

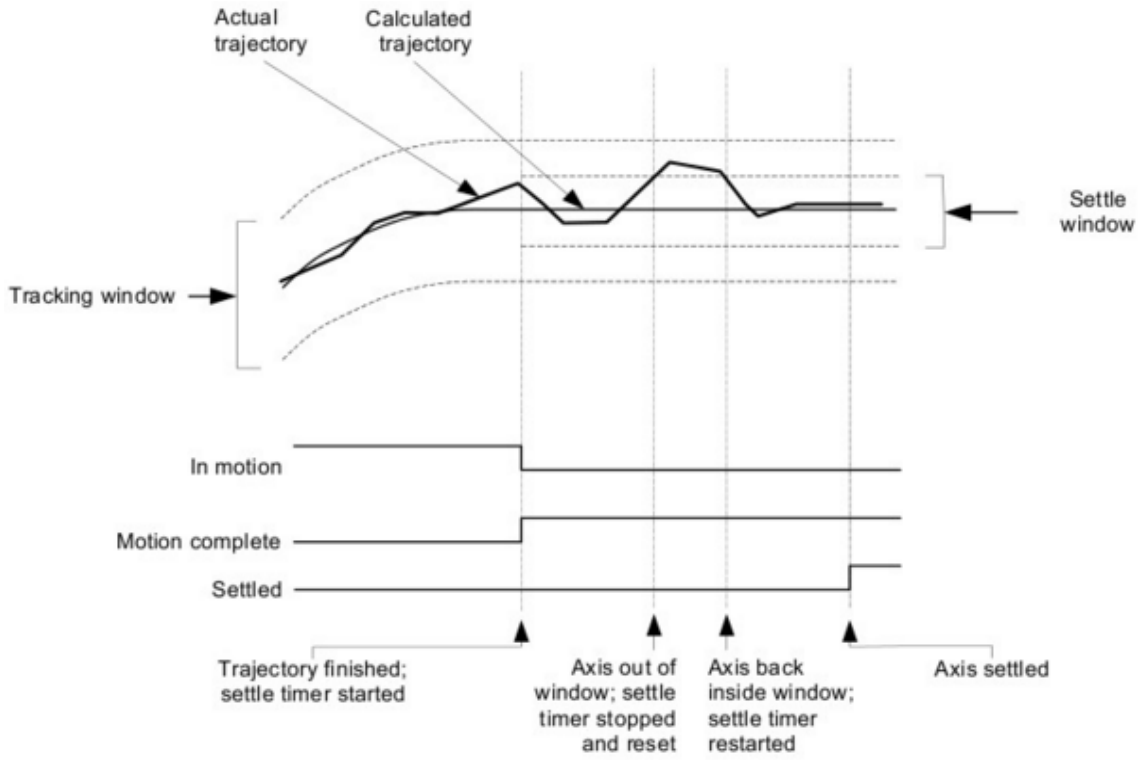

Abbildung 20: Settle Window

-o-

Trace Capture

"Data Trace" ist eine mächtige Funktion diverse Parameter und Register der POSYS® kontinuierlich zu überwachen und im On-Board Speicher der POSYS® zu speichern. Die ermittelten Daten können anschliessend vom Host heruntergeladen werden in dem man die Kommandos für den Zugriff auf den On-Board Speicher verwendet. "Data Trace" ist eine Funktion um die Regelparameter zu optimieren indem Vergleichswerte und Sensordaten ermittelt und mit Sollwerten verglichen werden.

Bei der Datenerfassung oder Abarbeitung von Bewegungsprofilen aus dem On-Board Speicher, wird der POSYS® zusätzliche erhebliche Arbeit abverlangt. In Applikationen in denen hoch-performante Szenarien vorkommen (z.B. Einzel-Achsenanwendung mit sich pro Updaterate verändernden Parametern), kann es unter Umständen sinnvoll sein die Updaterate zu erhöhen.

Es gibt generell 2 verschiedene Möglichkeiten der "Data Trace"-Erfassung. Die eine Möglichkeit wird durch die POSYS® bewerkstelligt, die andere durch den Host. Der Host spezifiziert welche Parameter erfasst werden sollen, und auf welche Art die Erfassung vor sich gehen soll. Die POSYS® vollzieht den "Trace" und der Host kann die Daten zur Auswertung abrufen (einmal Erfassung). Es ist aber auch möglich eine kontinuierliche Erfassung vom Host laufen zu lassen, während die POSYS® selber weiter Daten erfasst (kontinuierliche Erfassung).

-o-

Trace Buffer Architektur

Diese mächtige Funktion der Datenerfassung beruht auf die zur Verfügungstellung eines externen Speicherbereiches auf welches sowohl vom Host als auch vom Motion Controller zugegriffen werden kann. Es ist als Dual Port RAM ausgelegt (gleichzeitige Schreib- und Lesezugriffe) und kann auch als Single Port RAM konfiguriert und verwendet werden.

Die Größe des RAM's ist fix. Es kann sowohl als Memory Mapped RAM (schneller, aber Adress-abhängig) konfiguriert werden als auch als IO Mapped Memory (etwas langsamer, dafür immer verfügbar).

Um einen "Trace" zu starten, müssen eine bestimmte Anzahl Parameter definiert werden. Diese sind nachfolgend gelistet:

Parameter	Beschreibung
Trace buffer	Der Host muss den Bufferspeicher für die Erfassung initialisieren. Die POSYS® stellt verschiedene Instruktionen zur Verfügung um den On-Board Speicher als Buffer zu initialisieren und erlaubt dadurch Start- und Endpunkte des On-Board Speichers für die Speicherung zu definieren.
Trace variables	Es gibt 28 verschiedene Variablen die gespeichert werden können, als da wären Istposition, Event Status Register, Schleppabstandsfehler usw. Der Anwender muss auswählen welche Variablen und von welcher Achse diese erfasst werden sollen.
Trace period	Die POSYS® kann in frei-programmierbaren Zyklen (einzelner Zyklus, in unregelmäßigen Abständen oder mit einer bestimmten Frequenz) Daten erfassen. Diese Variable muss angegeben werden.
Trace mode	2 verschiedene Trace-Modi stehen zur Verfügung, einmal oder kontinuierlich. Diese spezifiziert auf welche Art die Daten gespeichert werden und ob der Trace-Modus automatisch stoppt oder vom Host unterbrochen werden muss.
Trace start/stop conditions	Damit die Datenerfassung präzise und synchronisiert abläuft ist es möglich Start- und Stopp-Bedingungen zu definieren. Die POSYS® erlaubt die Überwachung der spezifizierten Konditionen und startet und stoppt die Erfassung automatisch ohne Hostintervention.

Tabelle 32: Trace Buffer Parameter

-o-

Der Trace Buffer

Die POSYS® kann On-Board Speicher in Datenbuffer organisieren. Jeder Buffer erhält eine numerische ID (Adresse). Der "Trace Buffer" muss immer ID (0) sein. Ehe die Parametererfassung verwendet werden kann, muss der Speicherbuffer 0 mit einer gültigen Adresse und Länge programmiert werden.

Die Größe des Datenbuffers definiert die maximale Anzahl möglicher zu erfassende Datenpunkte. Die maximale Größe des Datenbuffers ist wiederum nur durch die maximal zur Verfügung stehende physikalische Speicherkapazität des Systems begrenzt. Der adressierbare Speicherbereich erlaubt 32x16 K Dual-Port RAM zu adressieren die komplett für die Speicherung von Daten verwendet werden können.

Während die Datenerfassung läuft, ist es nicht möglich die Konfiguration für die Datenerfassung zu ändern. Falls ein Versuch unternommen wird die Basisadresse, -länge oder den "Write Pointer", die mit Buffer 0 assoziiert sind, während der Erfassung zu ändern, wird dieser Versuch ignoriert und ein Error Flag gesetzt. Jedoch ist es möglich den "Read Pointer" und "Read Data" des Trace Buffers zu ändern während die Erfassung läuft. Dies erlaubt den Buffer kontinuierlich leeren zu können, während die Erfassung weiterläuft und wieder frei zu machen für die Erfassung neuer Daten.

-o-

Der Trace Period

Das Erfassungssystem unterstützt ein konfigurierbares Periodenregister. Dieser Periodenregister definiert die Frequenz innerhalb derer Daten im Trace Buffer gespeichert werden. Die Frequenz wird in Zykluseinheiten definiert, wobei ein Zyklus die minimale Periode darstellt um eine Erfassung für alle Achsen durchzuführen.

Das Kommando SetTracePeriod definiert die Länge der Periode, das Kommando GetTracePeriod liefert den programmierten Wert zurück.

-o-

Trace Variablen

Wenn eine Erfassung läuft ist es möglich zwischen 1 und 4 Parameter im Speicher pro Zyklus abzulegen. Die 4 "Trace Variable Registers" werden verwendet um zu definieren, welche Parameter gespeichert werden. Verwenden Sie die folgenden Kommandos für die Konfiguration. Das Kommando SetTraceVariable definiert welche Parameter gespeichert werden sollen. Das Kommando GetTraceVariable liefert die programmierten Werte zurück.

Der Wert des Traceparameters der gesetzt und zurückgelesen wird durch die beiden vorangegangen Kommandos, definiert die Achse und die Art der Daten die gespeichert werden sollen. Das Format des Datenwortes ist wie folgt:

Bits	Name	Beschreibung
0–3	TraceAxis	Selektiert die Quellachse für die Parametererfassung
4–7	Reserved	Muss 0 sein
8–15	VariableID	Selektiert die zu speichernden Parameter

Tabelle 33: Trace Variablen und Achsenauswahl

Die unterstützten Parameter-ID-Werte sind:

ID	Name	Beschreibung
Bahngenerator		
2	Commanded Position	Die augenblickliche Sollpositionsvorgabe durch den Regelkreis
3	Commanded Velocity	Die augenblickliche Sollgeschwindigkeitsvorgabe durch den Regelkreis
4	Commanded Acceleration	Die augenblickliche Sollbeschleunigungsvorgabe durch den Regelkreis
Enkoder		
5	Actual Position	Die Istposition des Motors
6	Actual Velocity	Eine geschätzte Istgeschwindigkeit (kalkuliert durch einen simplen Low-Pass Filter).
9	Capture Register	Die augenblicklichen Inhalte des High Speed Capture Registers.
15	Phase Angle	Der gegenärtige Phasenwinkel (nur bürstenlose Servomotoren).
16	Phase Offset	Der gegenwärtige Phasenversatzwert (nur bürstenlose Servomotoren).
Position Loop		
1	Position Error	Differenz zwischen Ist- und Sollposition für die spezifizierte Achse.
10	Position Loop Integral	Der momentane Integralwert
11	Position Loop Derivative	Der derivative Wert der vom Regelkreis verwendet wird.
57	Position Loop Integral Contribution	Dieses Register enthält den Anteil des integralen Teils der PID-Schleife
64	Biquad 1 Input	Der Eingabewert für den Biquad 1 Filter
65	Biquad 2 Input	Der Eingabewert für den Biquad 2 Filter
Status Register		
12	Event Status	Die augenblicklichen Inhalte des Event Status Registers.
13	Activity Status	Die augenblicklichen Werte des Activity Status Registers.
14	Signal Status	Die augenblicklichen Werte des Signal Status Registers.
Kommutierung		
7	Active Motor Command	Drehmomentsollwert (Ausgabe durch den Servofilter/Regelkreis).
17	Phase A Command	Der Wert der gegenwärtig ausgegeben wird an die Motorwicklung 1.
18	Phase B Command	Der Wert der gegenwärtig ausgegben wird an die Motorwicklung 2. Nur gültig für 2- bzw. 3-Phasenmotoren.
19	Phase C Command	Der Wert der gegenwärtig ausgegben wird an die Motorwicklung 3. Nur gültig für 3-Phasenmotoren.
29	Phase Angle Scaled	Der Phasenwinkel von 0°-360° anstatt in Encoderzählwerten
Analoge Eingänge		
20	Analog input 1	Der zuletzt gelesene Wert am Analogeingang 0.
21	Analog input 2	Der zuletzt gelesene Wert am Analogeingang 1.
22	Analog input 3	Der zuletzt gelesene Wert am Analogeingang 2.
23	Analog input 4	Der zuletzt gelesene Wert am Analogeingang 3.
24	Analog input 5	Der zuletzt gelesene Wert am Analogeingang 4.
25	Analog input 6	Der zuletzt gelesene Wert am Analogeingang 5.

ID	Name	Beschreibung
26	Analog input 7	Der zuletzt gelesene Wert am Analogeingang 6.
27	Analog input 8	Der zuletzt gelesene Wert am Analogeingang 7.
Verschiedenes		
0	None	Zeigt an dass keine Daten für die Tracevariable selektiert wurden
8	Motion Controller Time	Die Zykluszeit (Einheiten in Servoloop Updateraten).

Tabelle 34: Unterstützte Variablen-IDs

Wird der Parameter einer Variablen auf Null eingestellt, werden auch automatisch alle nachfolgenden Parameter deaktiviert. Deswegen, wenn x-Parameter während jeder Traceperiode gespeichert werden sollen, muss die Tracevariable 0 -(X-1) verwendet werden um zu identifizieren welche Parameter gespeichert werden sollen und die Tracevariable X muss gleich 0 sein.

Zum Beispiel; angenommen die Ist- und die Sollposition der Achse 3 sollen während eines jeden Zyklus gespeichert werden. Die folgenden Kommandos finden Anwendung um die Tracevariablen zu konfigurieren:

SetTraceVariable 0, 0202h	setzt die Tracevariable 0 Parameter 2 (Sollposition) für die 3. Achse zu speichern.
SetTraceVariable 1, 0502h	setzt die Tracevariable 1 Parameter 5 (Istposition) für die 3. Achse zu speichern.
SetTraceVariable 2, 0002h	deaktiviere die Tracevariable 2 und darüber.

Tabelle 35: Trace Beispiele

-o-

Trace Modes

Während der Erfassung der Daten werden diese sequentiell im Tracebuffer gespeichert. Wenn das Ende des Tracebuffers erreicht worden ist, wird die Erfassung auf zweierlei Arten reagieren, je nachdem welcher Modus selektiert wurde.

Falls "einmal" selektiert wurde, wird die Erfassung stoppen wenn der Buffer voll ist.

Falls der "kontinuierliche Modus" selektiert wurde, wird die Erfassung wieder am Anfang des Buffers Daten speichern (wrap-around). In diesem Modus wird die diagnostische Erfassung nicht enden ehe die Bedingungen, wie mit Kommando SetTraceStop gesetzt, erfüllt werden.

Verwenden Sie das Kommando SetTraceMode um den Modus auszuwählen. Das Kommando GetTraceMode liefert den programmierten Modus zurück.

-o-

Trace Start/Stop Bedingungen

Das Kommando SetTraceStart wird verwendet um die Bedingungen zu setzen die den Erfassungsmechanismus in Gang setzen. Ein ähnliches Kommando (SetTraceStop) wird verwendet um die Bedingungen zu setzen welche den Erfassungsmechanismus stoppen. Beide, SetTraceStart und SetTraceStop, sind ein 16-bit Datenwort welches 4 codierte Parameter enthält:

Bit	Name	Beschreibung
0-3	Trigger Axis	Für Triggertypen anders als unmittelbar, bestimmt dieses Feld, welche Achse als die Quelle für den Trigger benutzt werden soll. Verwenden Sie 0 für Achse 1, 1 für Achse 2, usw.
4–7	Condition	Definiert die Art von Trigger die benutzt werden soll. Sehen Sie in der Tabelle unten für eine vollständige Auflistung von Triggertypen nach.
8–11	Trigger bit	Für Triggertypen die auf den Statusregister basieren, bestimmt dieses Feld welches Bit (0-15) des Statusregisters überwacht werden soll.
12–15	Trigger State	Für Triggertypen die auf den Statusregister basieren, bestimmt dieses Feld welcher Status (0 oder 1) des spezifizierten Bits einen Trigger veranlassen

Bit	Name	Beschreibung
		wird.

Tabelle 36: Trace Start/Stop Bedingungen

Das Feld für den Triggertyp muss eines der folgenden Werte beinhalten:

ID	Name	Beschreibung
0	Immediate	Dieser Triggertyp zeigt an, dass die Erfassung sofort startet/stoppt wenn als Typ das Kommando SetTraceStart/SetTraceStop verwendet wurde. Wenn dieser Trippertyp spezifiziert wurde, dann werden die Werte für Quellachse, Bitnummer und Bitstatus nicht verwendet.
1	Update	Die Erfassung startet/stoppt beim nächsten Update der spezifizierten Quellachse. Dieser Triggertyp verwendet nicht die Bitnummer oder Statuswerte.
2	Event Status	Der spezifizierte Bit im "Event Status Register" wird kontinuierlich überwacht. Wenn der Bit den definierten Zustand einnimmt (0 oder 1) startet (stoppt) die Erfassung.
3	Activity Status	Der spezifizierte Bit im "Activity Status Register" wird kontinuierlich überwacht. Wenn der Bit den definierten Zustand einnimmt (0 oder 1) startet (stoppt) die Erfassung.
4	Signal Status	Der spezifizierte Bit im "Signal Status Register" wird kontinuierlich überwacht. Wenn der Bit den definierten Zustand einnimmt (0 oder 1) startet (stoppt) die Erfassung.

Tabelle 37: Trigger Typen

-o-

Download von Trace Daten

Wenn die Erfassung einmal gelaufen ist und der Tracebuffer voll (oder teilvoll) mit Daten ist, können diese Daten, durch Verwendung der Standardkommandos um den Buffer zu lesen, durch den Host heruntergeladen werden.

Wenn eine Erfassung stoppt (entweder durch Setzen des entsprechenden SetTraceStop-Kommandos oder weil das Ende des Tracebuffers erreicht wurde) wird der Lesezeiger des Tracebuffer's auf den ältesten Wert gesetzt. Falls kein Wrap-around stattfand, dann wird dies die Adresse 0 sein. Falls ein Wrap-around stattfand, wird der Lesezeiger auf den Speicherbereich gesetzt der von der nächsten Erfassung mit Daten überschrieben wurde.

Dieser Wert kann verwendet werden um die Anzahl der ReadBuffer Kommandos zu bestimmen um den kompletten Inhalt des Tracebuffer's herunter zu laden. Weil der Lesezeiger automatisch auf das älteste Datenwort gesetzt wird wenn die Erfassung endet und weil der Lesezeiger automatisch hochzählt und umbricht während Daten gelesen werden, ist das Lesen des kompletten Inhaltes des Tracebuffer's so einfach wie das zuweisen von x-ReadBuffer Kommandos (wobei x der Wert der zurückgeliefert wird vom Kommando GetTraceCount). Während jeder "Trace Period" wird jede "Trace Variable" benutzt einen 32-Bit Wert im Tracebuffer zu speichern. Daher, wenn Daten aus dem Buffer gelesen werden, wird der 1. Wert der gelesen wird der Variablen 1 zugeordnet, der 2. Wert der Variablen 2, usw.

Zu jeder Zeit kann das Kommando GetTraceCount verwendet werden um die 32-bit Datenwörter, die im Speicher gespeichert wurden, zu lesen. Beides, die Länge des Tracebuffers und die Anzahl der "Trace Variables" hat direkten Einfluss auf die Anzahl der zu erfassenden Anzahl Daten, die gespeichert werden können. z.B. wenn der Tracebuffer auf 1000 Datenwörter (32-Bit) begrenzt wurde und 2 Variablen initialisiert sind (Variable 0 und 1), dann können bis zu 500 Datensätze gespeichert werden. Jedoch wenn 3 Variablen (0, 1 und 2) erfasst werden, können nur 333 Datensätze gespeichert werden. In diesem Fall wird das übrige Datenwort die erste Variable von der 334. Erfassung speichern.

Wenn der Tracemodus kontinuierlich abläuft, dann wird die 334. Erfassung das 1. Datenwort im letzten verfügbaren Speicher des Tracebuffer's speichern, das 2. und 3. Datenwort werden entsprechend an den Plätzen 0 und 1 abgelegt. In diesem Fall werden die beiden ersten Worte der ersten Erfassung mit den 2 letzten Worten der 334. Erfassung überschrieben. Wenn die Erfassung gestoppt wird zeigt der Lesezeiger auf das älteste Datenwort im Buffer. Dieses Datenwort muss aber nicht das 1. Datenwort der Erfassung sein.

Daher wird empfohlen, dass die Länge des Tracebuffers definiert wird als eine multiple gerade Nummer der Anzahl von Tracevariablen die verwendet werden. Dies würde sicher stellen, dass der Leseindex auf das 1. Wort in einem kompletten Erfassungssatz zeigt, egal ob der Tracemodus einmal oder kontinuierlich abläuft. Die einfachste Lösung wäre es sicher zu stellen, dass die Tracebufferlänge eine multiple Gerade des 12-fachen ergibt (weil jede Anzahl von Variablen, 1, 2 , 3 oder 4, durch 12 teilbar ist).

-o-

Datenerfassung Starten

Hier ist eine Zusammenfassung der Datenerfassungsmöglichkeiten für den Anfang:

- *Spezifizieren Sie welche Daten gespeichert werden sollen.* Das Kommando SetTraceVariable (bis zu 4) wird verwendet um zu definieren welche Daten pro "Trace Period" erfasst werden sollen. Stellen Sie sicher, dass Variablen 0 - (x-1) verwendet werden, wenn X Variablen gespeichert werden sollen und Variable x gleich 0 ist. Die Variablen müssen in einem kontinuierlichen Modus verwendet werden. z.B. wenn 2 Variablen erfasst werden sollen, dann verwenden Sie die Variablen 1 und 2. Bei 3 Variablen 1, 2 und 3. Die erste Variable welche als none definiert wurde (siehe Kommandobeschreibung: SetTraceVariable) wird als die zuletzt zu erfassende Variable yngenommen. Falls die Variable 1 als none definiert wurde und Variable 2 wurde als actual position definiert, dann werden keine Variablen erfasst, da die erste Variable mit none definiert wurde und somit das Ende der Variablenliste angibt. Falls vier Variablen erfasst werden, muss none nicht noch gesondert angegeben werden.

- *Trace Buffer definieren.* Mit den Kommandos SetBufferStart und SetBufferLength werden die Position im On-Board RAM und die Menge des RAM, die für die Erfassung der Daten bereitgestellt werden soll, definiert. Die Trace Daten werden in einem Bufferbereich gespeichert mit ID = 0. Seien Sie auch vorsichtig nicht mehr als den tatsächlich zur Verfügung stehenden physikalischen Speicher zu definieren. Bedenken Sie auch dass SetBufferStart und SetBufferLength mit Werten definiert werden von 32 Bit Wortgröße.

- *Trace Period definieren.* Das Kommando SetTracePeriod erlaubt die Intervalle zu definieren in denen Daten erfasst und gesammelt werden sollen.

- *Definieren Sie den "Trace Mode".* Falls die Erfassung aufgrund eines Ereignisses stattfinden soll, dann sollte die "einmal"-Erfassung Verwendung finden. Dies würde es erlauben den kompletten Buffer mit Daten zu füllen und kann mit dem Eintreten des Ereignisses beginnen (Kommando SetTraceStart). Als Alternative, falls die Erfassung aufgrund eines Ereignisses stoppen soll (Kommando SetTraceStop), dann sollte die kontinuierliche Erfassung eingestellt sein. Dies würde das System dazu veranlassen kontinuierlich die Daten zu erfassen bis die Stop-Bedingung erfüllt wurde. Bis zu diesem Punkt würden die Daten, die bis zu dem Stop-Ereignis erfasst wurden, im Buffer gespeichert. Vorher erfasste Daten können wieder überschrieben worden sein, falls die Stop-Bedingung nach einem Wrap-around eintritt und der zur Verfügung gestellte Speicherbereich bereits mit neuen Daten beschrieben wurde.

- *Stoppmethode definieren (falls gewünscht).* Falls ein spezielles Ereignis die Erfassung beenden soll, dann sollten sie mit dem Kommando SetTraceStop programmiert werden. Jedoch, wenn die Erfassung angehalten werden soll wenn der Buffer voll ist ("einmal"-Erfassung), dann ist es nicht notwendig eine Stop-Bedingung zu programmieren. Es ist auch erlaubt mit dem Kommando SetTraceStop eine laufende Erfassung sofort zu stoppen.

- *Erfassung starten.* Das Kommando SetTraceStart kann verwendet werden eine Erfassung sofort beginnen zu lassen (Triggertyp: sofort). Als Alternative kann ein Ereignis spezifiziert werden, welches die Erfassung starten lassen soll.

-o-

Host Interrupts

Mit Interrupts ist es möglich den Host auf spezielle Ereignisse aufmerksam zu machen ohne die Notwendigkeit kontinuierlich oder durch Polling die Ereignisregister zu überwachen. Die POSYS® stellt diese Möglichkeit als eine Form des Hostinterrupt zur Verfügung.

Das eigentliche Signal welches generiert wird um eine Interruptbedingung anzuzeigen, hängt vom Kommunikationsmodus ab. Für die Kommunikation via ISA-Bus wird ein physisches Signal auf der Karte erzeugt. Bei der Kommunikation via CANBus erzeugen Hostinterrupts eine CAN-Nachricht. Bei der seriellen Kommunikation werden keine Hostinterrupts versendet.

Die Ereignisse die einen Hostinterrupt auslösen sind die selben die den Bits im "Event Status Register" zugewiesen wurden - hier noch einmal zur Erläuterung aufgelistet:

Bit	Ereignis	Passiert wann?
0	Motion complete	Das Profil erreicht seinen Endpunkt, oder die Bewegung wurde anderweitig zu einem Stop veranlasst.
1	Position wraparound	Positionszählerüberlauf mit Addierung
2	Breakpoint 1	Breakpoint 1 Bedingung eingetreten
3	Capture received	Enkoder- bzw. Homeschalterindex erfasst.
4	Motion error	Der maximale Schleppfehlerabstand wurde für eine bestimmte Achse überschritten.
5	Positive limit	Positiver Endschalter angefahren
6	Negative limit	Negativer Endschalter angefahren
7	Instruction error	Hostkommando verursacht Fehler
8-10	n/a	
11	Commutation error	Der Indextakt steht nicht im Einklang mit der aktuellen Phase
12-13	n/a	
14	Breakpoint 2	Breakpoint 2 Bedingung eingetreten

Tabelle 38: Host Interrupts

Bei der Benutzung einer 16-Bit Maske, kann der Host ein oder alle dieser Bits bedingen, um eine Unterbrechung zu verursachen. Diese Maske wird gesetzt mit dem Kommando SetInterruptMask. Der Wert der Maske kann zurückgelesen werden mit dem Kommando GetInterruptMask. Die Bitpositionen der Maske entsprechen den Bitpositionen des "Event Status Register". Wenn eine 1 in der Maske gespeichert wird, wird eine 1 in dem entsprechenden Bit des "Event Status Register" einen Interrupt auslösen. Jede Achse unterstützt seine eigene Interruptmaske. Dies erlaubt für jede Achse eine unterschiedliche Interruptbedingung zu setzen.

Die POSYS® fragt immer wieder das "Event Register" und die Interruptmaske ab um festzustellen ob ein Interrupt ausgelöst wurde oder nicht. Wenn ein Interrupt ausgelöst wurde, wird das Hostinterrupt-Signal aktiv.

An diesem Punkt kann der Host auf das Interrupt reagieren (obwohl die Ausführung der momentanen Hostanweisung, inklusive des Transfers aller damit verbundenen Datenpakete abgeschlossen sein sollte), aber es ist nicht unbedingt erforderlich.

Da es für mehr als eine Achse möglich ist, konfiguriert zu werden, Interrupts gleichzeitig zu generieren, stellt die POSYS® das Kommando GetInterruptAxis zur Verfügung. Dieses Kommando liefert einen bitmaskierten Wert mit einem gesetzten Bit für jede gegenwärtig eine Unterbrechung generierende Achse zurück. Bit 0 wird gesetzt wenn Achse 1 unterbricht, Bit 1 wird für Achse 2 gesetzt, etc. Wenn kein Interrupt ausgelöst wurde, dann sind keine Bits gesetzt.

Um ein Interrupt zu bearbeiten, können normale POSYS®-Kommandos verwendet werden. Die Kommandos die vom Host gesendet werden um den Interrupt zu behandeln, hängen von der unterbrechenden Bedingung ab, jedoch muss als Mindestes das unterbrechende Bit im "Event Status Register" mit dem Kommando ResetEventStatus zurückgesetzt werden. Falls dies nicht geschieht, würde augenblicklich der selbe Interrupt ausgelöst, nachdem die Interrupts wieder aktiviert sind.

Sobald der Host die Bearbeitung des Interrupts abgeschlossen hat, sollte ein ClearInterrupt-Kommando gesendet werden, um den Interruptanschluss zu löschen und wieder frei zu machen. Anmerkung: falls ein weiterer Interrupt zur Bearbeitung ansteht wird die Interruptleitung nur für den momentanen Interrupt wieder frei gemacht und erneut belegt.

Falls der Controller mit dem Host via CAN kommuniziert, werden Motion Controller Interrupts die über den Parallelbus (ISA-Bus) gesendet werden als CAN Benachrichtigungen versendet. Weitere Informationen hierzu finden Sie im Kapitel „Controller Area Network: CAN“.

Im Folgenden ist eine typische Sequenz von Interrupts und Hostreaktionen. In diesem Beispiel hat eine Achse einen Endschalter in der positiven Richtung angefahren, die ein Endschalterereignis und einen abrupten Stopp verursachte. Der abrupte Stopp verursacht einen Schleppabstandsfehler. Nehmen wir an, dass diese Ereignisse alle mehr oder weniger gleichzeitig eintreten. In diesem Beispiel ist die Unterbrechungsmaske (interrupt mask) für diese Achse gesetzt worden, so dass entweder Schleppabstandsfehler oder Endschalterereignisse einen Interrupt verursachen.

Ereignis	Hostaktion
Schleppabstandsfehler und Endschalterereignis generieren einen Interrupt	Sendet GetInterruptAxis Kommando
Eine Bitmaske, die alle unterbrechenden Achsen identifiziert, wird von der POSYS® zurückgeliefert. Dieser Wert identifiziert eine Achse	Sendet GetEventStatus Kommando und ermittelt daß die Flags für Schleppabstandsfehler und Endschalterereignis gesetzt sind. Sendet daraufhin ein ResetEventStatus Kommando um beide Bits zurückzusetzen. Setzt mit dem Kommando SetMotorMode ON

Ereignis	Hostaktion
als die Interupt verursachende Achse.	wieder die Achse in den Closed Loop Mode zurück. Ein ClearInterrupt-Kommando setzt die Interruptleitung zurück.
POSYS® setzt Schleppabstandsfehlerbit zurück und deaktiviert Hostinterruptleitung	Generiert eine negative Bewegung um vom Endschalter herunter zu fahren.
Motor bewegt sich vom Endschsalter. Activity Status Endschalter Bit ist gelöscht.	Keine

Tabelle 39: Ereignis und Hostaktion bei einem Hostinterrupt

Am Ende dieser Sequenz werden alle Zustandsbits gelöscht, die Interruptleitung ist inaktiv, und es stehen keine weiteren Interrupts unbearbeitet an.

-o-

Hardware Control Signals

Hardware Control Signals

Es gibt eine Anzahl Signale für jede Achse der POSYS® die verwendet werden können, um die Aktivität der POSYS® mit Ereignissen von außerhalb zu koordinieren. In diesem Abschnitt werden wir diese Signale beschreiben. Dies sind die bi-direktionalen Endschalter, der AxisIn-Eingang und der AxisOut-Ausgang. Diese Signale existieren für jede Achse der POSYS®.

-o-

Das AxisOut-Signal

Jede Achse verfügt über einen generellen achsen-spezifischen Ausgang, welches programmiert werden kann auf jeden Zustand der zugewiesenen Bits im "Event Status Register, "Activity Status Register" oder "Signal Status Register" zu agieren. Eine oder mehrere Bits können ausgegeben werden basierend auf den SelectionMask und die Signale können mit der SenseMask programmiert werden „high“ oder „low“ aktiv zu sein. Hierfür wird das Kommando SetAxisOutMask verwendet und das Kommando GetAxisOutMask liefert den programmierten Zustand zurück.

Die überwachten Bit(s) in einem dieser Register können der gleichen oder einer anderen Achsen als der Achse dessen AxisOut-Signal überwacht wird, sein. Diese Funktion ist nützlich für die Ansteuerung externer Peripherie durch Ausgabe von Hardwaresignalen. Zum Beispiel erlaubt es das AxisOut-Signal zu steuern, wenn Bedingungen wie "Schalte AxisOut wenn Motion Complete Bit ist „low“, oder wenn Kommutierungsfehlerbit „high“ ist" auftreten. Ob das AxisOut-Signal active „high“ oder active „low“ schaltet, kann mit dem Kommando SetSignalSense bestimmt werden.

Es ist möglich den AxisOut-Ausgang als ein direktes unter Hostkontrolle laufendes Software-programmiertes Ausgangsbit zu benutzen. Dies kann bewerkstelligt werden indem man null selektiert als Register-ID mit dem Kommando SetAxisOutSource und indem man den inaktiven Ausgangsstatus mit dem Kommando SetSignalSense als active „high“ oder active „low“ je nach Bedarf einstellt.

-o-

Der AxisIn-Eingang

Jede Achse hat einen für allgemeine Zwecke verwendbaren achsenspezifischen Eingang, welches mit dem Kommando GetSignalStatus gelesen werden kann und um auch automatisch Ereignisse zu erfassen, wie z.B. durch einen Signalübergang erzeugte Bewegungsänderung (Stop, Start, Geschwindigkeitsänderung usw.) aufgrund der Verwendung von Breakpoints.

Es sind keine weiteren speziellen Kommandos notwendig um AxisIn zu aktivieren oder deaktivieren.

-o-

Analoge Eingänge

Die POSYS® verfügt über 8, für allgemeine Zwecke verwendbare analoge Eingänge. Diese 8 Eingänge sind mit einer internen Schaltung verbunden, die die analogen Signale in digitale Signale mit 10 Bit Auflösung umwandelt. Die Konvertierung geschieht kontinuierlich und passiert auf allen 8 Kanälen. Die Konvertierung benötigt gerade mal 21,6 µSek. (basierend auf der normalen Motion Controller Clock Frequenz).

Mit dem Kommando ReadAnalog wird der zuletzt konvertierte Wert gelesen. Der Wert der mit diesem Kommando zurückgeliefert wird, resultiert daraus indem der konvertierte 10-Bit Wert 6 Bit nach rechts versetzt wird. Die Analogdaten sind für allgemeine Zwecke gedacht und werden nicht vom Motionprozessor für irgendwelche Kalkulationen verwendet.

Diese analogen Eingänge sind geeignet die POSYS® mit Joysticks zu verbinden um z.B. eine Achse per Handbetrieb in eine bestimmte Position zu bringen, oder für Teaching-Funktionen. Hierbei können in Verbindung mit der Trace-Funktion die Positionen abgespeichert und anschliessend vom Host ausgelesen werden. Damit wären komplette, aufwendige Konturen erfass- und anschliessend abfahrbar.

-o-

The Synch Pin - Multiple Card Synchronization

Die POSYS® 182x-B unterstützen die Synchronsation der internen Zykluszeit mehrerer Controller. Dies erlaubt die Synchronisation von Start/Stop und die Modifizierung von Bewegungsabläufen über mehrere Controller hinweg wo präzises Timing erforderlich ist. Dies kann notwendig sein, weil die Regler sich in größerer Entfernung befinden oder die Applikation mehr als 4 Achsen von synchronisierter Bewegung erfordern. In der üblichsten Konfiguration wird ein Regler als ein Master ausgewiesen und alle anderen Regler werden in den Slave-Modus gesetzt.

Multi-Karten-Synchronisation ist nicht möglich sobald eine Achse im Schrittmotormodus betrieben wird. Schrittmotorkarten sind grundsätzlich nicht in diesem Modus zu betreiben.

Der Eingang/Ausgangs Status von diesem IO und seiner Aufgabenstellung werden mit dem Kommando SetSynchronizationMode gesetzt.

Encoding	Modus	Beschreibung
0	Disabled	In diesem deaktivierten Modus ist die Verbindung als Eingang konfiguriert und nicht in Verwendung. Dies ist der Standardmodus nach einem Reset oder nach dem Einschalten.
1	Master	Im Master-Modus gibt der Controller einen Synchronisationstakt aus der von den Slave-Controllern oder anderen Geräten verwendet werden kann um den internen Arbeitszyklus mit dem Master zu synchronizieren.
2	Slave	Im Slave-Modus ist der Controller derart konfiguriert dass das eingehende Synchronisationssignal den internen Arbeitszyklus synchronisiert mit dem Eingangssignal.

Tabelle 40: Encoding für den Synchronizationsmodus

Wenn man mehrere Regler synchronisiert, müssen die folgenden Regeln beachtet werden:

- Die Servoloopupdaterate aller Regler muss gleich sein. Z.B. wenn eine 4-Achsen-POSYS® und eine 2-Achsen POSYS® miteinander synchronisiert werden sollen, muss mindestens der Größere der beiden möglichen niedrigsten Werte eingestellt sein.
- Nur eine Karte im Verbund kann als Master fungieren. Z.B. wenn 3 Controller zu einem Verbund zusammen geschaltet wurden, kann nur eine Karte als Master, die anderen beiden Karten müssen als Slaves konfiguriert sein.
- Im Verbund müssen zuerst die Slaves definiert werden und dann die letzte Karte als Master. Dies stellt sicher, dass die Zykluszeiten der Slaves präzise auf die Zykluszeit des Master eingestellt werden.

Der Master sendet alle 50 µs ein Taktsignal an die Slaves. Die Applikationen des Anwenders können dieses Signal auch zur Synchronisation weiterer externer Peripheriegeräte verwenden.

-o-

Enkoderschnittstelle

Enkoderschnittstelle

Die POSYS® 18xx-B unterstützt 2 verschiedene Arten der Positionsrückmeldung: inkrementale Enkoderrückmeldung und SSI Absolutwertgeber mit 10, 12, 13 und 25 Bit Auflösung (Single- und Multiturn). Die Art der Rückmeldung ist programmierbar. Des weiteren kann die Enkoderschnittstelle auf Wunsch für den Betrieb mit EnDat®- oder BiSS®-fähigen Enkodern im bi-direktionalen Modus konfiguriert werden.

Um die Art der Enkoderrückmeldung zu wählen wird das Kommando SetEncoderSource verwendet. Um die gewählte Einstellung abrufen zu können verwenden Sie das Kommando GetEncoderSource.

-o-

Inkrementaler Enkodereingang

Die inkrementale Rückmeldung liefert 2 Rechtecksignale zurück. A und B und einen optionalen Index, der normalerweise anzeigt wann der Motor eine volle Umdrehung zurückgelegt hat. Das Offset der beiden Signale A und B beträgt 90° (Abbildung 21).

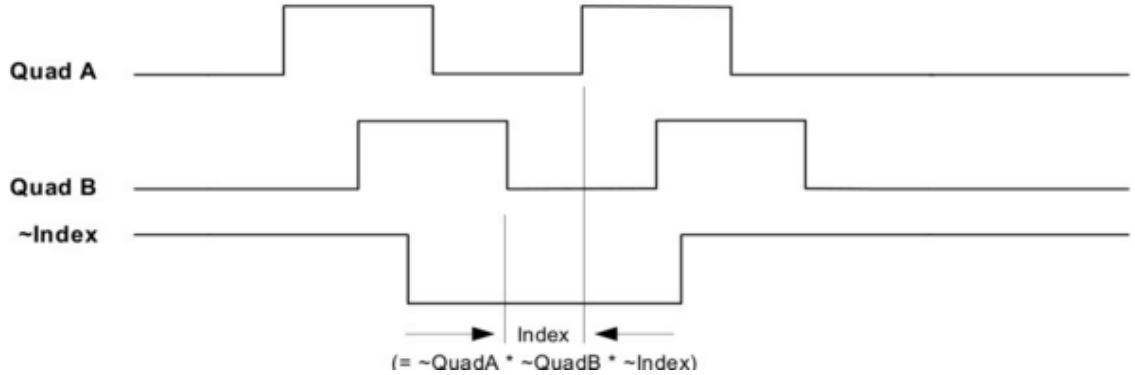

Abbildung 21: 4-fach Auswertung des inkrementalen Enkodereinganges

Damit die Enkoderposition von der POSYS® korrekt interpretiert werden kann, sollte das A-Signal dem B-Signal vorangehen wenn der Motor sich in positiver Richtung bewegt (Abbildung 21). Wenn der Motor sich in negativer Richtung dreht, soll das B-Signal dem A-Signal vorausgehen. Wegen des 90° Offsets erhält die POSYS® 4 Takte für eine voll erfasste Phase für beide Signale, A und B.

Die POSYS® unterstützt auch die Verwendung invertierter Signale, was eine wesentlich höhere Signalsicherheit gewährleistet. Hierzu müssen für beste Signalqualität die Netzwerkwiderstände (RS604, RS608 und RS613) gesteckt sein.

-o-

Aktuelle Positionsregister

Die POSYS® überwacht kontinuierlich die Rückmeldung und akkumuliert einen 32-Bit Positionswert, genannt "Actual Position". Nach dem Einschalten ist die aktuelle Position gleich null. Die akuelle Position kann allerdings mit dem Kommando SetActualPosition und AdjustActualPosition neu definiert werden und mit dem Kommando GetActualPosition kann die aktuelle Position abgefragt werden.

Im Allgemeinen bei Verwendung von Inkrementalgebern, wird die aktuelle Position kurz nach dem Einschalten gesetzt, indem eine Prozedur aufgerufen wird mit deren Hilfe eine physikalisch definierte Hardwareposition als Referenz angefahren wird.

-o-

Digitale Filterung

Alle Enkoder- als auch der Index- und die Homeinputs sind digital gefiltert um die Verlässlichkeit zu erhöhen. Der Filter erfordert, dass ein gültiger Übergang nur dann akzeptiert wird, wenn es den neuen Status (low oder high) für 175 ns beibehält. Dies soll sicherstellen, dass kurze Störsignale nicht als Enkodersignale fehlinterpretiert werden.

Obwohl dieses digitale Filterungsschema die allgemeine Verlässlichkeit erhöht kann es trotzdem notwendig sein zusätzliche Techniken einzusetzen um die höchstmögliche Verlässlichkeit zu erreichen, wie z.B. der Einsatz von Differential Line Drivers/Receivers, oder analoge Filterung. Ob diese zusätzlichen Filterungen notwendig sind, hängt vom System ab und ob und wie viele Störsignale auftreten mögen.

Neben der normalen Filterung durch den Motion Controller verfügt die neue POSYS 1800-B auch über einen zusätzlichen Baustein mit dem die Signale für die frei-programmierbaren digitalen Eingänge und für die Endschalter mit einem zusätzlichen zeit-basierten Filter gefiltert werden können. Dieser Filter verfügt über einen Dipschalter (SW600) mit dem man die Zeitkonstante einstellen kann. Bitte lesen Sie im Kapitel „1800-B Hardware Handbuch“ die Details hierzu durch.

-o-

High-Speed Position Capture

High-Speed Position Capture

Jede Achse der POSYS® 1800-B unterstützt einen High Speed Positionserfassungsregister um die aktuelle Achsenposition zu erfassen, getriggert durch ein externes Signal. Eines von zwei Signalen kann als Eingangstrigger verwendet werden um entweder das Indexsignal oder die Home Position zu triggern.

Diese beiden Eingänge differieren in der Art wie die Erfassung erkannt wird. Falls das Indexsignal getriggert wird, wird das Signal gesetzt wenn das A, B und Indexsignal eine bestimmte Position einnehmen (definiert im Signal Sense Register der mit dem Kommando SignalSenseRegister verwendet wird). Falls die Home Position gewählt wurde die es zu erfassen gilt, dann muss nur dieses Signal einen bestimmten Zustand einnehmen um getriggert zu werden.

Die voreingestellten Werte für das A, B, Index- und Homezustandssignal im "Signal Sense Register" ist 0, was bedeutet dass diese Signale "active low" sind. In diesem Zustand, wenn Index getriggert werden soll, wird eine Erkennung dann stattfinden, wenn A, B und Index alle "low" sind. Jede Änderung in der Zustandsmaske (Signal Sense Mask) wie das Signal interpretiert werden soll, würde dementsprechend eine Änderung in der Erkennung des Triggerzustandes zur Folge haben.

Das Kommando SetCaptureSource definiert ob das Index- oder das Homesignal für die Positionserfassung getriggert werden soll. Das Kommando GetCaptureSource liefert den programmierten Wert zurück.

Wenn eine Erfassung getriggert wurde, werden die Inhalte der aktuellen Position in das "Position Capture Register" transferiert und der Erfassungserkennungsindikator (Bit 3 des Event Status Register) wird gesetzt. Mit dem Kommando GetCaptureValue kann man den "Capture Register" lesen. Der "Capture Register" muss gelesen werden, ehe eine neue Erfassung stattfinden kann. Das Lesen des "Capture Register" setzt den Trigger zurück, was bedeutet dass eine weitere Erfassung stattfinden kann. Wie für alle Event Status Register Bits kann das Kommando ResetEventStatus verwendet werden um den Positionserfassungsindikator zu löschen.

-o-

Absolutenkodereingang (SSI, BiSS, EnDat)

Absolutenkodereingang (SSI, BiSS, EnDat)

Systeme die keine inkrementalen Rückmeldungen bieten, aber stattdessen ein digitales Binärwort, für die bietet die POSYS 1800-B nun standardmäßig auch die Möglichkeit SSI Absolutwertgeber anzuschließen. Optional und nach Rücksprache mit POSYS Motion Control, ist es auch möglich bi-direktionale Absolutwertgeber mit BiSS oder EnDat Schnittstelle anzuschließen. Für den Anschluß von SSI Absolutwertgebern bitte sehen Sie im Hardware-Handbuch unter Kapitel „SSI Absolut Enkoder Schnittstelle: CON804“ bzgl. Details nach.

-o-

Multi-Turn Systeme

Zusätzlich zur Unterstützung von Positionserfassungen über einen vollen numerischen Bereich, unterstützt die POSYS® auch sogenannte Multi-turn Systeme. Die Parallelenkoderwerte die eingelesen werden, werden kontinuierlich überprüft und ein Überlauf automatisch erkannt, sowohl in positiver als auch in negativer Richtung. Die Werte werden hinzu addiert.

Bei Verwendung dieses "virtuellen" Multi-Turn Zählers speichert die POSYS® kontinuierlich die Position in einem 32-Bit Wert. Natürlich, wenn die Rückmeldeeinheit keinen Überlauf produziert (non Multi-Turn System), wird ein 16-Bit Wert zurückgeliefert.

Wenn der Motor sich in positiver Richtung bewegt, erhöht sich der Wert bis zu einem Maximum. An diesem Punkt des Überlaufes wird von 0 aus weitergezählt und wieder kontinuierlich hochgezählt. Das Gleiche geschieht wenn sich der Motor in negativer Richtung bewegt. Der Wert bei dem der Zähler der Rückmeldeeinheit überläuft, wird "Devices's Modulus" genannt und sollte mit dem Kommando SetEncoderModulus gesetzt werden.

Anmerkung: das Kommando SetEncoderModulus verwendet als Parameter einen Wert der die Hälfte des Modulus beträgt.

Z.B. ein Motor verwendet einen 12-Bit Resolver als Rückmeldeeinheit. Der Modulus beträgt 4.096, der Wert der mit SetEncoderModulus daher gesetzt werden soll, beträgt 2.048. Jedesmal wenn der Binärwert seinen höchsten Wert erreicht und wieder auf seinen kleinsten "zurückspringt", erkennt dies die POSYS® und der Wert zu dem der Überlauf passiert, wird zur aktuellen Position hinzu gezählt mit Werten größer als 4.096 oder kleiner als 0.

Für Systeme die einen Positionszähler mit einem Modulus kleiner als die maximale Anzahl Striche/Umdrehung verwenden (Enkoderauflösung pro Umdrehung), muss der Wert für die Anzahl Striche pro Umdrehung gleich dem Wert für die Positionszählergröße sein. Z.B. wenn ein rotierendes Laserinterferometer verwendet wird, welches ein 16-Bit Wert ausgibt, aber 16.777.216 Zähler pro Umdrehung ausgibt, muss der Zähler pro Umdrehungswert 32.768 (216/2) sein.

Mit den Parallelwortrückmeldeeinheiten ist eine High Speed Positionserfassung nicht möglich. Diese für diesen Zweck gedachten Signale (A, B, Index und Home) werden in diesem Modus nicht verwendet. Nichsdestotrotz ist die Erfassung über diese Schnittstelle sehr schnell und zuverlässig. Mit unserer integrierten SSI-Schnittstelle wird jede Achse bis zu 20.000x pro Sekunde gelesen.

Trotzdem können für nichtpositionäre Erfassungsfunktionen, die den "Signal Status Register" verwenden, diese Signale normal verwendet werden. Z.B. können diese Bits gelesen (Kommando GetSignalStatus) oder zum Triggern von Unterbrechungspunkten verwendet werden etc.

Die High-speed Positionserfassung steht bei Achsen die mit Absolutwertgebern verwendet werden nicht zu Verfügung. Die Eingänge für Index, Home und für die inkrementalen Enkodereingänge können nicht verwendet werden.

-o-

Motorausgabesignale

Motorausgabesignale

Die POSYS 1800-B Motion Controller verfügen über ein Motor Output Module welches kontrolliert welche Art Motorausgabesignale generiert und an den Verstärker ausgegeben werden. In diesem Kapitel wird erläutert, wie Sie den Motor Output Module aktivieren und deaktivieren, wie man die Motorart auswählt und wie man auswählt und die Schnittstelle konfiguriert zu anderen Verstärkern mit unterschiedlichen Schnittstellentechniken.

-o-

Motor Output Module deaktivieren

Es gibt eine Reihe von Gründen warum es gelegentlich wünschenswert sein kann das Motor Output Module zu deaktivieren. Für weitere Informationen siehe bitte „Aktivieren und Deaktivieren von Control Modules“ für einen allgemeinen Überblick über das Deaktivieren/Aktivieren von Control Modules. Die Hauptgründe ein Motor Output Module zu deaktivieren ist entweder um bei Bedarf Sicherheit zu gewährleisten, oder für die Systemkalibrierung. Um das Motor Output Module zu deaktivieren wird das Kommando SetOperatingMode verwendet. Welcher Wert gesetzt wurde kann mit dem Kommando GetOperatingMode verifziert werden.

Falls das Motor Output Module deaktiviert ist, wird ein „Null“-Kommando an den Motor oder an jede Phase des Motors, falls es ein bürstenloser Servomotor oder Mikroschrittmotor mit mehreren Phasen ist, ausgegeben. Je nachdem welches Ausgabeformat gewählt wurde, wird dieses „Null“-Kommando auf unterschiedliche Arten ausgegeben. Die folgenden Kapitel geben hierüber weitere Informationen über die Motorausgabesignalart.

Die Deaktivierung des Motor Output Modules kann oder kann auch nicht einen Motor sofort stoppen. Die Deaktivierung hat eigentlich den Effekt den Motor in einen Freilauf zu versetzen. Der Motor kann stoppen, weiter driften, oder sogar beschleunigen falls eine externe Kraft auf den Motor wirkt. Die standardmäßige Einstellung für das Motor Output Module ist „aktiviert“.

-o-

Aktivieren des Motor Output Modules

Ein vorher deaktiviertes Motor Output Module kann auf verschiedene Arten wieder aktiviert werden. Falls es deaktiviert wurde mit dem Kommando SetOperatingMode, dann kann das gleiche Kommando verwendet werden es wieder zu aktivieren. Falls das Modul deaktiviert wurde aufgrund eines automatisch ausgeführten, sicherheits-bedingten Ereignisses, muss dann das Kommando RestoreOperatingMode ausgeführt werden.

Egal wie das Modul wieder aktiviert wird, zu dem Zeitpunkt ab dem die Aktion ausgeführt wurde, werden die Motorsollwertvorgaben für jede Phase unverzüglich an den Verstärker ausgegeben. Man muss Vorsicht walten lassen beim aktivieren dieses Moduls und darauf achten dass der Unterschied zwischen Soll- und Istpositionen so gering wie möglich ist um abrupte Motorbewegungen zu vermeiden. Des weiteren ist es empfehlenswert, falls weitere Module (Current Loop, Position Loop und Bahngenerator) aktiviert werden sollen, diese alle zur gleichen Zeit zu aktivieren, dabei sicherstellend dass interne Current Loop oder Position Loop Werte (z.B. Integralwerte) zum Zeitpunkt der Aktivierung Nullwerte aufweisen.

-o-

Motortyp

Die POSYS 1800-B Positioniersteuerungen unterstützen eine Reihe von Motoren. Das sind z.B. bürstenbehaftete und bürstenlose Servomotoren, Mikroschritt- und Schrittmotoren. Im Falle von bürstenlosen Servomotoren und Mikroschrittmotoren werden zudem je nach Motorausgabesignal 2- als auch 3-Phasenmotoren unterstützt. Die folgende Tabelle gibt hierüber Aufschluß:

Produkt	Unterstützte Motorversionen
POSYS 182x-B	DC bürstenbehaftet bürstenlose DC, 2 Phasen bürstenlose DC, 3 Phasen Mikroschritt, 2 Phasen Mikroschritt, 3 Phasen Schritt
POSYS 185x-B	Schritt

Tabelle 41: Unterstützte Motortypen (Anmerkung: POSYS 185x-B unterstützt auch alle Motoren wie die POSYS 182x-B falls sie mit digitalen Treibern verwendet wird die Unterstützung bietet für Takt- & Richtungssignaleingang für den Motorsollwert)

Für die POSYS 182x-B gilt, dass für jede Achse die Motortype ausgewählt werden muss und dieser Wert dafür hergenommen wird um eine Reihe von Standardeinstellungen zur Verfügung zu stellen und zu bestimmen welche Control Modules aktiviert werden und welche Art von Signalausgabe zum Verstärker bereitgestellt wird.

Während des Einschaltens liest der Motion Controller das Motorkonfigurationswort an einer bestimmten Adresse auf seinem externen Bus um die Standardmotortypeinstellungen jeder Achse zu bestimmen. Bei der POSYS 182x-B antwortet die Logik auf diese Leseoperation aufgrund der Steckbrückenkonfiguration CON805 & CON806. Diese Steckbrückenkonfiguration definiert bestimmte Ausgänge für entweder um die erste Gruppe von Motoren oder Schrittmotoren ansteuern zu können (siehe vorangehende Tabelle).

Die Auswahl des direkten Motortyps in der ersten Gruppe wird durch ein weiteres Kommando SetMotorType durchgeführt. Welcher Wert gesetzt wurde, kann mit dem Kommando GetMotorType abgefragt werden. Das Kommando SetMotorType sollte als eines der ersten Kommandos ausgeführt werden nach dem Einschalten, falls eine Serie von weiteren Kommandos die Motor-spezifisch sind, ausgeführt werden sollen und die Art der Verstärkeranbindung definieren. Dies ist auch deswegen notwendig da eine Reihe von Werten und Einstellungen erst vorgenommen werden können, wenn der Motortyp definiert wurde. Für weitere Informationen verweisen wir hier auf das POSYS 1800-B Programmierhandbuch.

-o-

Motorausgabeformat

Verschiedene Methoden stehen bei der POSYS® zur Verfügung mit entsprechenden Treibern und Verstärkern verbunden und verwendet zu werden, jedoch variieren die Methoden bei den verschiedenen Motorversionen und Modellen.

POSYS 18xx-B	Motortyp	Phasen pro Achse	Verfügbare Ausgabeformate
POSYS 182x-B	DC bürstenbehaftete oder DC bürstenlos Servomotoren mit externe Kommutierung	1	Sign/magnitude PWM 50/50 PWM Parallel DAC
	DC bürstenlose Servomotoren mit interne Kommutierung	2 oder 3	Sign/magnitude PWM* 50/50 PWM Parallel DAC
	Mikroschrittmotor	2 oder 3	Sign/magnitude PWM * 50/50 PWM Parallel DAC
POSYS 182x-B, 185x-B	Schrittmotor	1	Pulse & Direction

Tabelle 42: Motorsollwertformatausgabe

* nur wenn 2 Phasen selektiert sind. Dieses Format ist nicht für 3-Phasen Motoren verfügbar.

Sign/magnitude PWM, 50/50 PWM, und DAC Ausgangssignale verwenden einen Vorzeichen definierten numerischen Sollwert, allerdings in verschiedenen Formaten. Alle dieser Formate kodieren das Signal dahingehend, dass der numerische Wert einem Drehmoment- bzw. Geschwindigkeitssollwert entspricht, welche die POSYS® an den Verstärker ausgibt.

Takt & Richtung arbeitet auf eine fundamental andere Art als die beiden vorhergenannten Methoden, weil bei dieser Methode keine Möglichkeit besteht eine PID-Regelung zu verwenden. Stattdessen unterstützt Takt & Richtung Schrittmotortreiber die entweder einen Takt für die positive Richtung oder einen Takt für die negative Richtung erhalten. Der Verstärker selbst ist voreingestellt auf die Motorverstärkung für die es meistens verschiedene Einstellmöglichkeiten gibt. An dieser Stelle noch mal der Hinweis, dass Takt- & Richtungssignale im Allgemeinen

dafür verwendet werden Schrittmotoren zu steuern, aber auch verwendet werden kann mit Schrittmotor-ähnlichen Servosystemen die dieses Signal unterstützen.

Das Kommando SetOutputMode definiert welches der zur Verfügung stehenden Ausgabemethoden verwendet wird. Das Kommando GetOutputMode liefert den programmierten Wert zurück.

-o-

Sign / Magnitude PWM

In diesem Modus werden 2 Ausgänge verwendet um den Motor anzutreiben. Ein Ausgang liefert das impuls-breiten-modulierte Signal welches von 0% bis 100% sein kann. Dieses Signal steht für die maximale Magnitude für den gewünschten Drehmomentsollwert. Ein Signal auf diesem Ausgang bedeutet, dass auf die Motorwicklung eine Spannung anliegen soll. Ein zweiter Ausgang definiert ob das Motorausgangsignal für eine positive oder für eine negative Drehrichtung sorgen soll.

In diesem Modus ist der Ausgang ein Teil von 2048. Die Auflösung des Motorbefehlsregisters ist skaliert auf Werte von -32768 bis +32767 um den PWM-Ausgabebereich abzudecken.

Z.B. wenn das Ausgabesignal für eine Achse der POSYS® +12345 sein soll, dann wird das Richtungssignal „high“ sein und das impuls-breiten-modulierte Signal wird mit folgendem Zyklus arbeiten: 2048*12345/32768 = 771,56 = 772. Dies bedeutet, dass das impuls-breiten-modulierte Signal für 772 Zyklen high sein wird und für die restlichen 1276 Zyklen low. Für die negative Richtung (-12345) würde das impuls-breiten-modulierte Signal das Gleiche bleiben aber das Richtungssignal wäre stattdessen low.

-o-

50 / 50 PWM

In diesem Modus wird nur ein Ausgang verwendet pro Motorausgang oder Motorphase. Dieser Ausgang trägt das von einem variablen Arbeitszyklus abhängigen impuls-breiten-modulierte Signal, dass dem vorhergehenden beschriebenen Signal sehr ähnelt. Im Gegensatz bedeutet aber ein Zyklus von 50% einen Drehmomentsollwert von 0 darstellt und den Motor in der Position halten soll. Negative Fahrtrichtungen werden damit bewerkstelligt, dass der Zyklus weniger als 50% beträgt und positive Richtungen werden gefahren wenn der Zyklus einen Wert höher als 50% hat. In diesem Modus bedeutet 100% Ausgabewert, volle positive Richtungsbewegung und 0% volle negative Richtungsbewegung.

Z.B. wenn das Ausgabesignal welches der Controller ausgibt für eine bestimmte Phase +12345 ist, dann sieht die Ausgabeformel so aus: 1024+1024*12345/32768 = 1409,78 = 1410. Für 1410 Zyklen ist das impuls-breiten-modulierte Signal high und für die verbleibenden 638 Zyklen low. Wenn der gewünschte Wert aber -12345 ist 1024+24*-12345/32768 = 638,2 = 638, dann ist das impuls-breiten-modulierte Signal für 638 Zyklen high und low für die verbleibenden 1410 Zyklen.

50/50 PWM-Signale werden mit 2 verschiedenen Verstärkerversionen verwendet. Wenn ein bürstenloser Permanentmagnetmotor angetrieben werden soll ist das impuls-breiten-modulierte Signal mit Half Bridge Treiber angeschlossen. Wenn ein DC bürstenbehafteter Motor verwendet wird, wird ein Half Bridge Verstärker benutzt, allerdings ist das Magnitudensignal von der Half Bridge immer eingeschaltet, und der Magnitudenausgang der POSYS® wird mit dem Vorzeicheneingang von der Half Bridge verbunden. Diese alternative Methode eine Half Bridge anzusteuern ist ab und zu nützlich in Situationen in denen man während der Abbremsrampe mit Rück-EMF rechnen muss, das die standardmäßigen Vorzeichenmagnitudensysteme benutzt.

Abbildung 22 zeigt ein typisches 50/50 PWM Ausgabeformat.

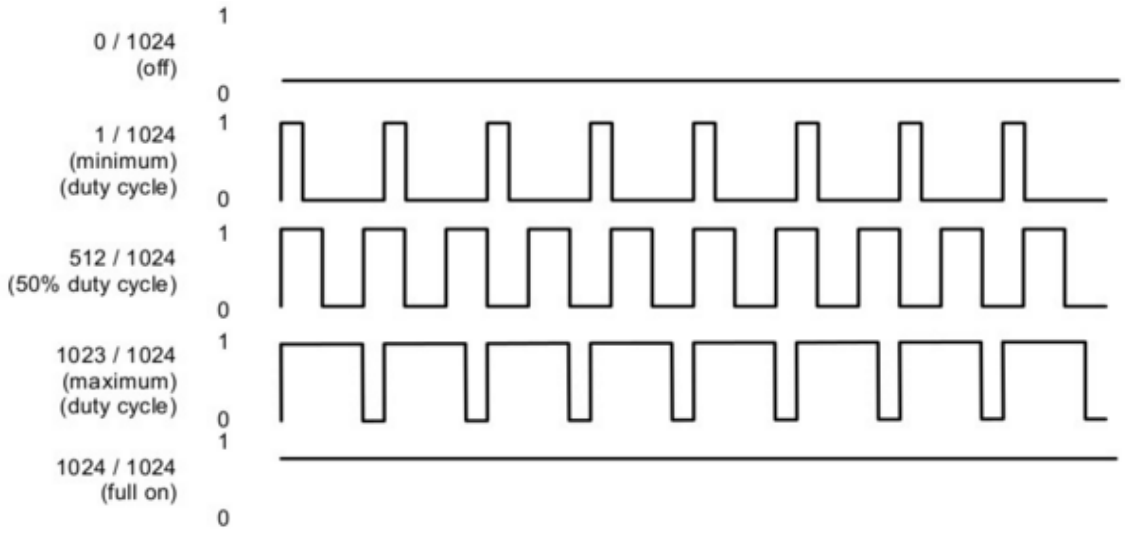

Abbildung 22: 50/50 PWM encoding

-o-

16 Bit DAC Signalausgabe

In diesem Modus wird das Ausgabesignal direkt auf den Peripheriebus der POSYS® ausgegeben. Das digitale Signal wird mit Hilfe eines seriellen DAC in ein analoges Signal umgewandelt. Die POSYS® schreibt dieses Signal nacheinander auf jede aktivierte Achse im Rythmus der Servoloop Updaterate. Z.B. bei einer 4-Achsen POSYS® 1824-B Karte, bei der alle 4 Achsen aktiviert sind und die konfiguriert sind ein analoges Signal auszugeben, wird die DAC-Ausgabe für jede Achse einmal alle 100 µs neu ausgegeben und zwar in Paaren zu je 50 µs.

Für 1- bzw. 2-phasige Motoren wird ein DAC-Ausgang für jede Phase benutzt. Für 3-phasige Motoren werden nur 2 DAC-Ausgänge verwendet. Die dritte Phase wird immer ein analoges Signal sein gleich -1*(P1+P2) wobei P1 das Ausgabesignal für Phase 1 ist und P2 das Ausgabesignal für Phase 2. Im Bedarfsfall kann dieses dritte Phasensignal bei der Benutzung eines invertierenden und kummulierenden Verstärkers in der externen Schaltungsanordnung realisiert werden. Im allgemeinen ist dies nicht notwendig, da die Mehrzahl von 3-Phasen Standardverstärkern zwei Phasen akzeptiert und intern das dritte konstruieren.

Die Auflösung des Analogsignals ist 16 Bit. Dieser Wert wird um 8000h versetzt ausgegeben, so wird ein Wert von 0 dem negativsten Wert entsprechen. Ein Wert von 8000h entspricht einem Ausgabewert von Null, und ein Wert von 0FFFFh entspricht dem positivsten Ausgabewert.

-o-

Einstellung der PWM-Frequenz

Falls die impuls-breiten-modulierte Signalausgabe verwendet wird, kann es unter Umständen sinnvoll sein die Ausgabefrequenz anzupassen. Mit dem Kommando SetPWMFrequency wird dies bewerkstelligt. Welche Frequenz eingestellt ist, kann mit dem Kommando GetPWMFrequency abgefragt werden.

Anmerkung: Dieses Kommando steht für Karten mit erweiterter PID-Funktionalität (Piezo Funktionalität) nicht zur Verfügung.

Produkt	PWM Frequenzauswahlmöglichkeiten
POSYS 182x-B	20kHz / 10 Bit (1/1,024) Ausgabeauflösung. 80kHz / 8 Bit (1/256) Ausgabeauflösung.
POSYS 182x-B Piezo	20kHz / 10 Bit (1/1,024) Ausgabeauflösung. (fester Wert)

Tabelle 43: Setzen der PWM Frequenz

Die Wahl einer dieser Frequenzen hängt von Motortyp und Anwendung ab. Für die POSYS 1824-B und POSYS

1824-B Piezo sollte 20kHz für alle Motortypen und PWM-Modi ausgewählt werden, außer wenn ein Low Pass Filter verwendet wird, um aus den PWM-Signalen analoge Signale zu erzeugen. In diesem Fall sollten 80kHz ausgewählt werden (außer POSYS 1824-B Piezo).

-o-

Multi-Phasen Motoranbindung

Die allgemeinen Konzepte der PWM oder analogen Ausgangssignale gelten sowohl für DC Servomotoren als auch Multi-Phasen-Motoren (bürstenlose Gleichstrommotoren, Schrittmotoren). Abhängig von der Wellenform und der gewählten Motorleistung (PWM, DAC), werden entweder zwei oder drei Ausgangssignale pro Achse vom Motion Controller zur Verfügung gestellt. Dies ist in der folgenden Tabelle aufgeführt.

Motortyp	Motorausgabemodus	Anzahl Ausgabesignale & Name
3-phase	PWM5050	3 (A, B, C)
3-phase	PWMSign/Mag	2 (A, B)
3-phase	DAC	2 (A, B)
2-phase	PWM5050	2 (A, B)
2-phase	PWMSign/Mag	2 (A, B)
2-phase	DAC	2 (A, B)

Tabelle 44: Multi-Phasen Motoranbindung

Für DC Servomotoren mit einer einzigen Phase gilt, dass das PWM oder analoge Signal eine einzelne Motorwicklung bestromt. Für Multi-Phasen Motoren unterscheidet sich das Anbindungsschema leicht abhängig davon ob PWM oder analoge Ausgangssignale verwendet werden.

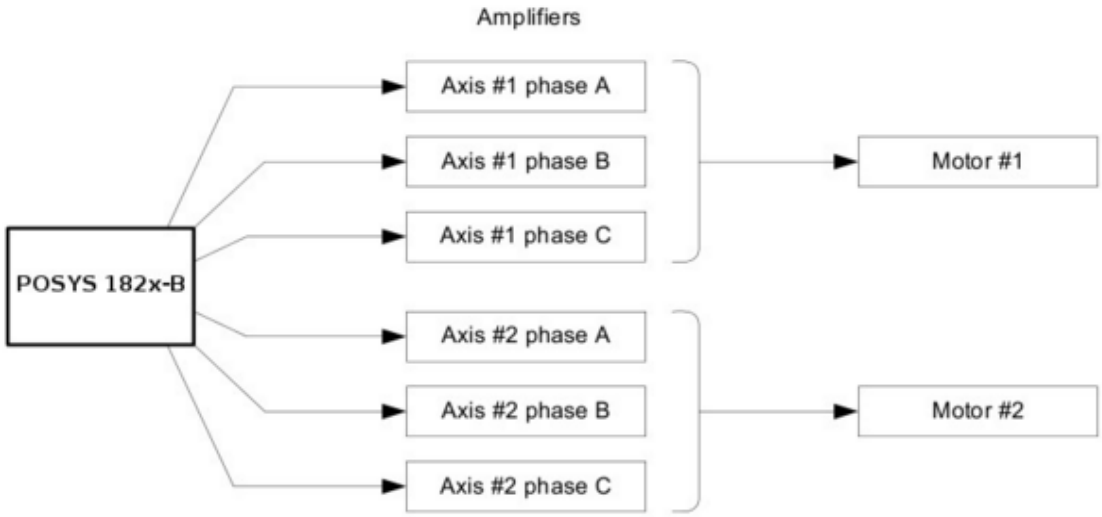

Abbildung 23: EC Servomotoren (PWM Mode) Anschlußschema

Abbildung 23 illustriert eine typische Verstärkerkonfiguration für ein 3-Phasen EC Servomotor in PWM-Modus. In dieser Konfiguration gibt der Motion Controller auf 3 Phasen PWM-magnitude Signale pro Achse aus. Diese Signale gehen direkt auf drei half-bridge-artige Verstärker.

Abbildung 24 zeigt eine typische Verstärkerkonfiguration für ein 3-Phasen EC-Servomotor welches über die analoge Signalausgabe des Motion Controllers geregelt wird.

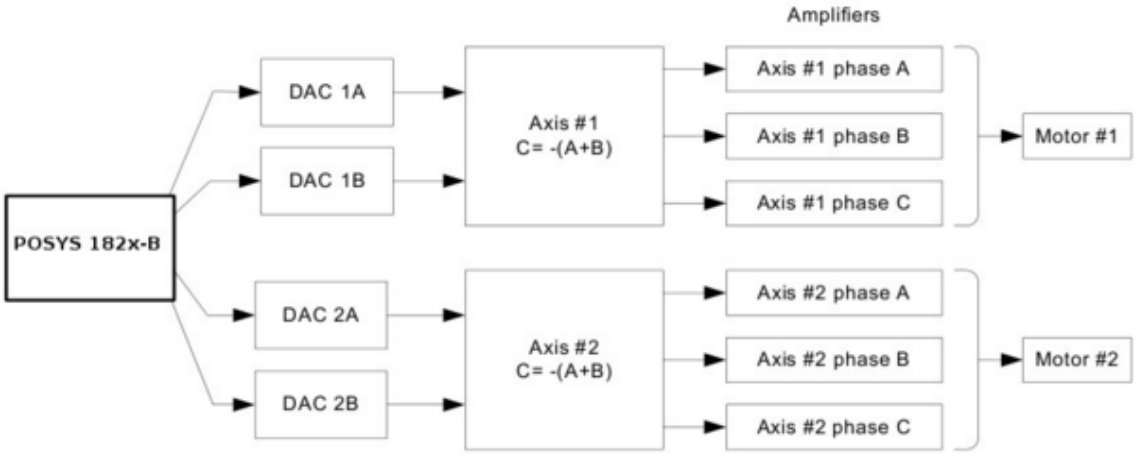

Abbildung 24: EC-Servomotor (DAC Mode) Anschlußschema

Bei der analogen Signalausgabe muss das digitale Wort vom Motion Controller erst in eine Spannung umgewandelt werden. Das wird erreicht durch ein externen seriellen DAC. Zwei DAC Kanäle (A- und B-Phasen) werden pro Achse benötigt. Um die dritte Phase (C-Phase) für einen EC-Servomotor zu generieren, wird die Summe aus den A und B Signalen gegen C negiert (C = -(A+B)). Dies wird mit einer op-amp Schaltung bewerkstelligt. Zusätzlich, falls Drehmomentkontrolle erwünscht ist, werden die drei Ausgabesignale so ausgegeben, dass der Strom der letztendlich durch die Motorwicklungen fließt in der Summe Null (0) ergibt.

-o-

Multi-Phasen Motorkommandointerpretation

Wie für DC Bürstenmotoren stellen die Befehle für die Multi-Phasen Motorphasen eine gewünschte Spannung oder Drehmomentsollwertvorgabe durch die Spule, und dieses Kommando kann positiv oder negativ sein. Im Gegensatz zu 1-phasigen Bürstenmotoren, sind die Wellenformen für Multi-Phasen Motoren sinusförmig, wobei das Ausmaß der Sinuskurve das gesamte gewünschte Motordrehmoment widerspiegelt, und der Winkel der Sinuskurve die Phasenanforderungen reflektiert um die Motorwicklungen korrekt zu erregen. Abbildung 25 zeigt die gewünschte Wellenform der Ausgangsspannung für eine einzelne Phase eines Multi-Phasen Motors.

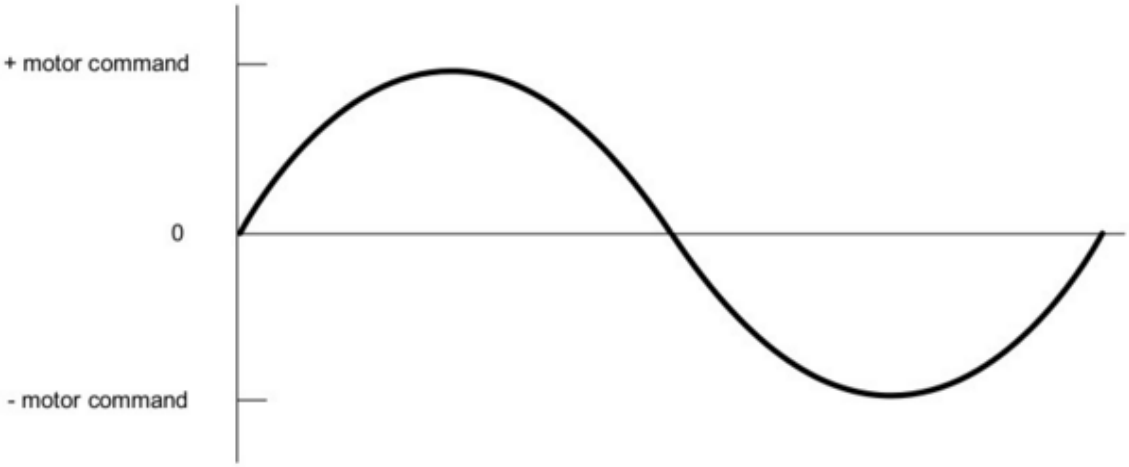

Abbildung 25: Sinus-förmige Motorsollwertausgabe

Die Wellenform ist um die 0V herum zentriert. Abhhängig davon welche Module aktiviert sind, ist das Ausmaß der Wellenform proportional zur Ausgabe des Position Loops, des Current Loops oder des Motor Command Registers.

Z.B. falls der Motion Controller über den DAC mit ±10V Ausgabebereich verbunden ist und am Motion Controller sind sowohl der Bahngenerator und das Position Loop Module deaktiviert und der Motor Command Register ist programmiert mit dem Wert 32767 (maximaler Ausgabewert; SetMotorCommand), dann während der Motor eine Umdrehung durch einen vollen elektrischen Zyklus ausführt, wird eine um die 0V zentrierte sinusförmige Wellenform ausgegeben, dessen maximaler und minimale Spannung einmal +10V und dann einmal -10V beträgt.

-o-

Takt- & Richtungssignalausgabe

Für Schrittmotoren werden diese Art Signale ausgegeben, um Treiber für Schrittmotoren oder digitale Verstärker für Servomotoren die dieses Format unterstützen, anzutreiben.

Jedes Taktsignal das der Motion Controller ausgibt, entspricht einem Inkrement für die zu initiierende Bewegung. Dieses Signal ist immer ein Rechtecksignal. Standardmäßig wird ein Taktsignal als ausgeführt angesehen sobald das Signal von „low“ auf „high“ wechselt. Das Signal entspricht nicht notwendigerweise einem 50%-igen Zyklus, jedoch sind die ansteigenden und fallenden Flanken des Rechtecksignals präzise abgestimmt. Um die Erkennung des Taktsignales umzukehren, verwenden Sie bitte das Kommando SetSignalSense. Weitere Informationen erhalten Sie auch im Kapitel „Signal Sense Mask“.

Das Richtungssignal ist mit dem Taktsignal synchronisiert. Standardmäßig bedeutet ein „high“-Signal in die positive Richtung zu verfahren, ein „low“-Signal bedeutet negative Verfahrrichtung. Dieses Signal kann mit dem Kommando SetSignalSense invertiert werden. Bitte beachten Sie: falls als Rückmeldung „internal Loop-back“ eingestellt ist, muss das „A“-Enkoder Signal ebenfalls invertiert werden um die Übereinstimmung zwischen Richtung und aktuelle Motorbewegung zu erhalten.

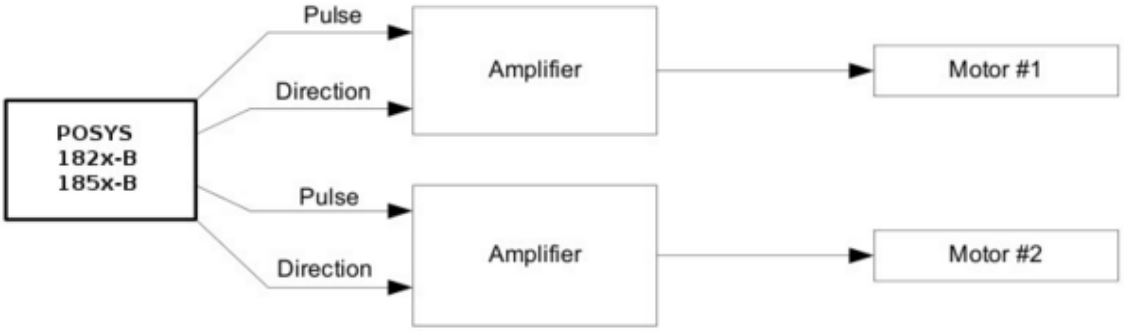

Abbildung 26: Schrittmotoranbindung

Die POSYS 18xx-B Motion Controller unterstützen seperate Taktratenmodi. Dies wird mit dem Kommando SetStepRange eingestellt. Die nachfolgende Tabelle zeigt die möglichen Werte und daraus resultierenden Taktraten an:

Kommando	Frequenzbereich der auszugebenden Takte
SetStepRange 1	0 to 4.98 M steps per second.
SetStepRange 4	0 to 622.5 K steps per second.
SetStepRange 6	0 to 155.625 K steps per second.
SetStepRange 8	0 to 38.90625 K steps per second.

Tabelle 45: Wählbare Taktraten

Die Bereiche geben die minimale und maximale Taktrate an die vom Motion Controller ausgegeben werden kann, wenn eingestellt auf einer der Werte. z.B. falls die gewünschte maximale Taktrate 200K Takte pro Sekunde beträgt, dann sollte die Einstellung SetStepRange 4 sein.

Für Voll- und Halbschrittapplikationen als auch Takt- & Richtungsapplikationen bei denen die maximale Geschwindigkeit voraussichtlich 38 kSchritte/Sek. nicht überschreitet, sollte SetStepRange *8* verwendet werden. Für höhere Geschwindigkeiten sollte einer der anderen Geschwindigkeitsbereiche verwendet werden.

Für jede Achse kann ein seperater Geschwindigkeitsbereich eingestellt werden. Welcher Geschwindigkeitsbereich eingestellt wurde kann mit dem Kommando GetStepRange überprüft werden.

Der Taktgenerator ist standardmäßig so eingestellt, dass ein Taktsignal als gegeben erscheint wenn das Rechtecksignal von „low“ auf „high“ wechselt. Manche Schrittmotorsysteme interpretieren ein Taktsignal erst als gegeben an wenn das Rechtecksignal von „high“ auf „low“ wechselt. In diesem Fall muss das Kommando SetSignalSense ausgeführt werden um die Interpretation des Taktsignales zu invertieren. Das Kommando SetSignalSense kann auch verwendet werden das Richtungssignal zu invertieren. Für weitere Informationen verweisen wir auf das Programmierhandbuch für die POSYS 1800-B und die Beschreibung des Kommandos SetSignalSense.

Für weitere Informationen in Bezug auf Schrittmotorsteuerung verweisen wir auf das Kapitel „Schrittmotorkontrolle“.

-o-

Mikroschrittmotorsignalausgabe

Die POSYS 182x-B unterstützt für Mikroschritttreiber die Ausgabe von DAC und PWM Signalen. Die Ausgabeart ist für jede Achse individuell einstellbar. Das Kommando um die gewünschte Ausgabeart zu selektieren ist SetOutputMode mit dem dazugehörigen Parameter welches die Ausgabemethode definiert die ausgegeben werden soll.

-o-

Motorausgangssignal Interpretation

Abbildung 27 zeigt die typische Wellenform für eine einzelne Phase der POSYS 182x-B. Jede Phase hat eine ähnliche Wellenform, aber die Phase des B-Kanals ist um 90° bzw. 120° (hängt ab von der gewählten Wellenform) versetzt im Vergleich zur Phase des A-Kanals.

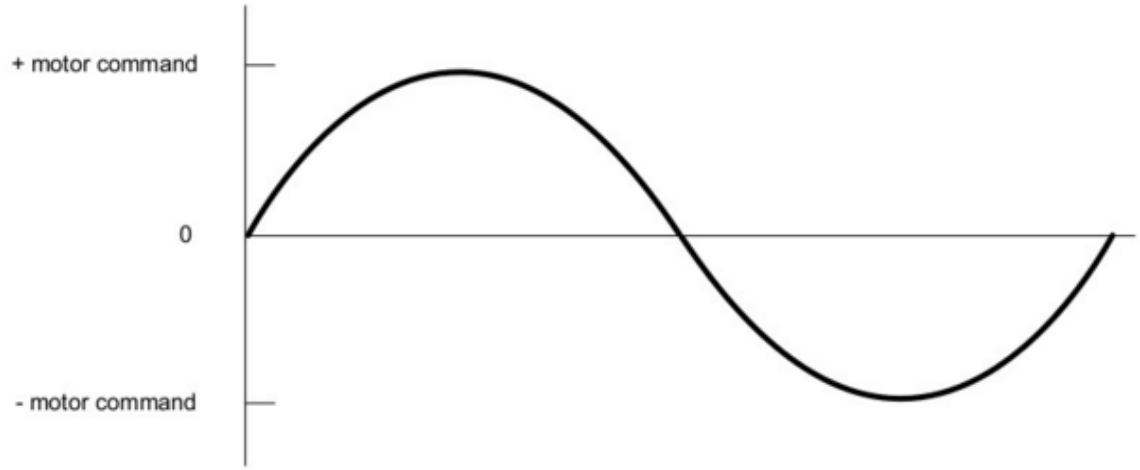

Abbildung 27: Typische Motorwellenform

Die Wellenform ist bzgl. des Ausgabewertes um die 0V zentriert. Der Maximalwert der ausgegegebenen Wellenform wird über das Motor Command Register kontrolliert (SetMotorCommand).

Z.B. falls der Motion Controller über den DAC mit ±10V Ausgabebereich verbunden ist und am Motion Controller sind sowohl der Bahngenerator und das Position Loop Module deaktiviert und der Motor Command Register ist programmiert mit dem Wert 32767 (maximaler Ausgabewert; SetMotorCommand), dann während der Motor eine Umdrehung durch einen vollen elektrischen Zyklus ausführt, wird eine um die 0V zentrierte sinusförmige Signalform ausgegeben, dessen maximale und minimale Spannung einmal +10V und dann einmal -10V beträgt.

-o-

Impuls-Breiten-Modulierte (PWM) Motorsignalausgabe

Das folgende Diagramm zeigt die typische Signalform eines Signals einer einzelnen Phase für die Mikroschrittversionen in PWM-Modus (sign/magnitude). Jede Phase gibt ein ähnliches wellenförmiges Signal aus, wobei die Signale um 90° bzw. 120° verschoben sind (je nachdem ob für 2-Phasen oder für 3-Phasen Schrittmotoren). Die Methode mit der diese Signale generiert werden, unterscheidet sich vom DAC digital word.

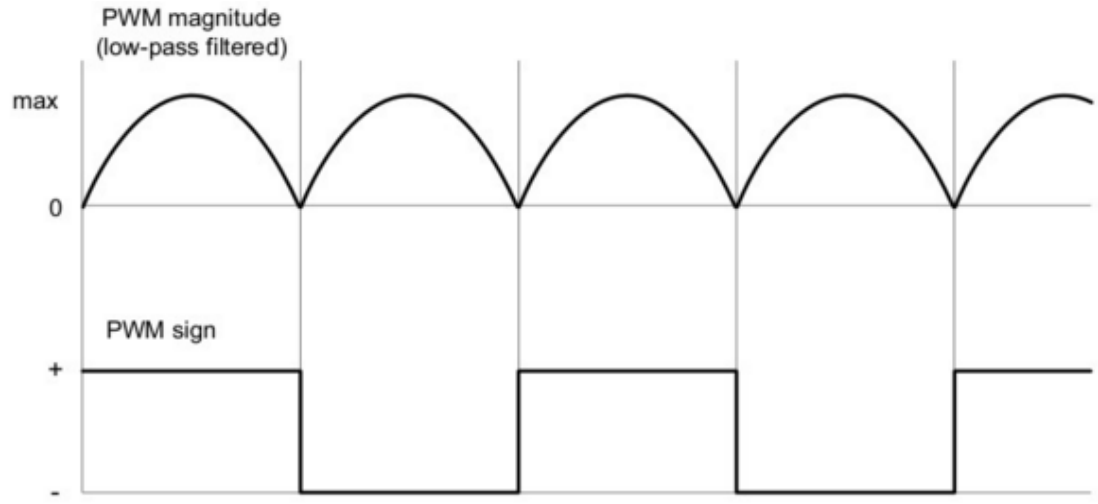

Abbildung 28: gefiltertes PWM sign/magnitude Signal

In diesem Diagramm wurde das PWM Magnitudensignal gefiltert um es von einem variablen digitalen Rechtecksignal in ein analoges Signal umzuwandeln.

Vor der Filterung entspricht das PWM Signal einem entsprechenden impuls-breiten-moduliertem digitalem Signal der gewünschten analogen Spannung. Der PWM-Zyklus hat eine Frequenz von 78,124 kHz. Die Auflösung ist 8 Bit bzw. entspricht 1/256.

-o-

Motortreiber-Konfigurationen

Abbildung 29 illustriert eine typische Treiberkonfiguration eines 2-Phasen Schrittmotors das entweder PWM oder DAC Signaleausgabemodi verwendet.

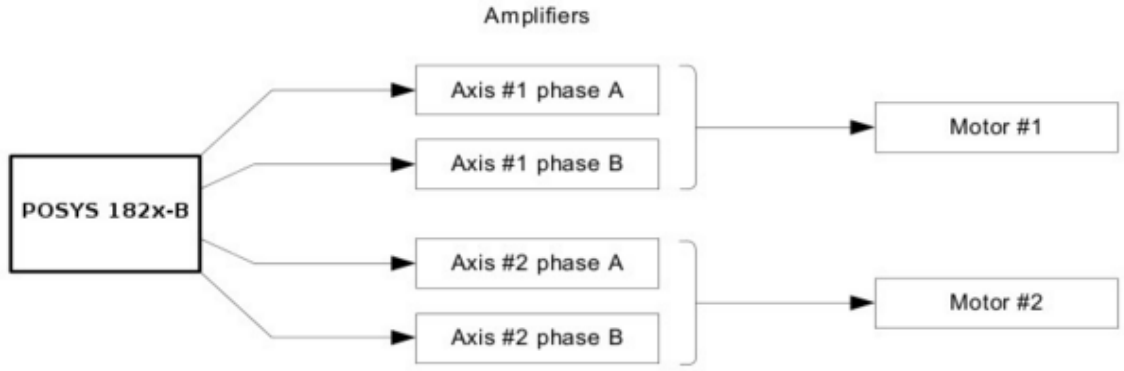

Abbildung 29: Typische Treiberkonfiguration für 2-Phasen Schrittmotor

Bei Verwendung eines DACs wird der digitale Motorausgabepegel für jede Phase gewöhnlich konvertiert in ein Analogsignal mit einem Wert zwischen -10V und +10V. Dieses Signal kann dann auf einen handelsüblichen DC-Servoverstärker ausgegeben werden (ein Verstärker für jede Phase) oder in irgendeinen linearen oder schaltenden Verstärker welches Drehmomentkontrolle ausübt und bipolare Ausgabe unterstützt.

In diesem Schema treibt ein Verstärker eine Phase des Schrittmotors an und der Controller generiert die erforderlichen sinusförmigen Signale für jede Phase um eine saubere und akkurate Bewegung zu gewährleisten.

Falls der PWM-Ausgabemodus verwendet wird, werden das Takt- und Richtungssignal für gewöhnlich mit einem H-Bridge-typischen Gerät verbunden. Für maximale Leistung sollte der Verstärker auf Drehmomentsollwert eingestellt sein. Dies minimiert die Stromverzerrung der Spule aufgrund von Induktivität und EMF. Obwohl es einige Methoden gibt die benutzt werden können, um Drehmomentsollwertskontrolle mit PWM zu erreichen, ist es eine übliche Methode das PWM-Signal durch einen Low Pass Filter zu schicken, dabei ein analoges Referenzsignal erstellend welches direkt mit dem Strom durch die Spule verglichen werden kann.

Diverse Verstärker sind verfügbar die kompatibel sind mit diesen Eingangssignalen. Diese Verstärker benötigen einen analogen Referenzeingang (low-passed PWMMag Signal vom Controller) als auch ein Bit mit Vorzeichen (PWMSign vom Controller). Der Verstärker generiert intern einen Drehmomentsollwert anhand eines PWM Drive Schemas.

Das nachfolgende Diagramm zeigt ein solches Drive Schema:

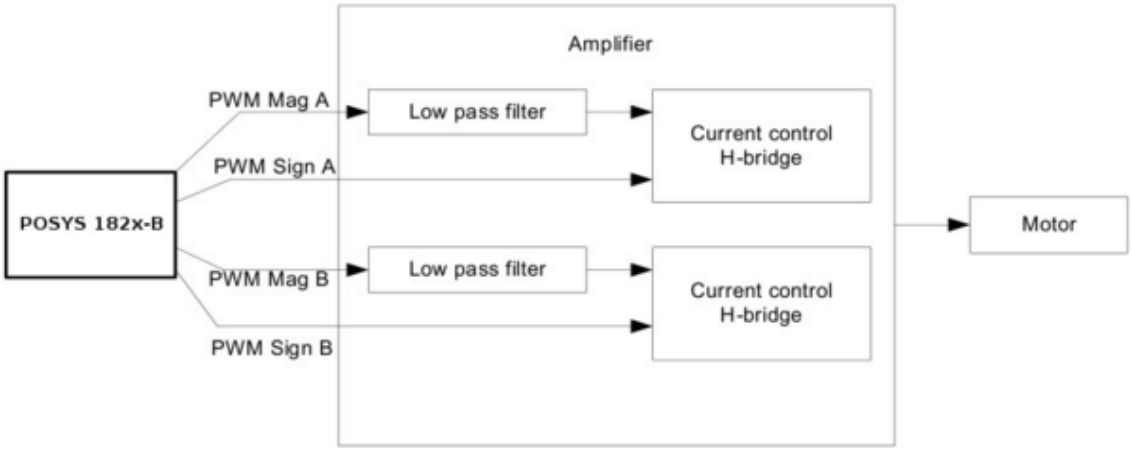

Abbildung 30: Typische Verstärker Strom-kontrollierte Konfiguration

Im Vergleich zur DAC-Methode hat die PWM-Variante in Verbindung mit diesem Verstärkerschema den Vorteil hohe Leistung zu erbringen bei einem Minimum an externen Teilen.

Nachfolgend ist eine typische Verstärkerkonfiguration mit den Mikroschrittversionen der POSYS® in Verbindung mit einem 3-Phasen Schrittmotor oder für einen bürstenlosen Servomotor mit 3 Phasen.

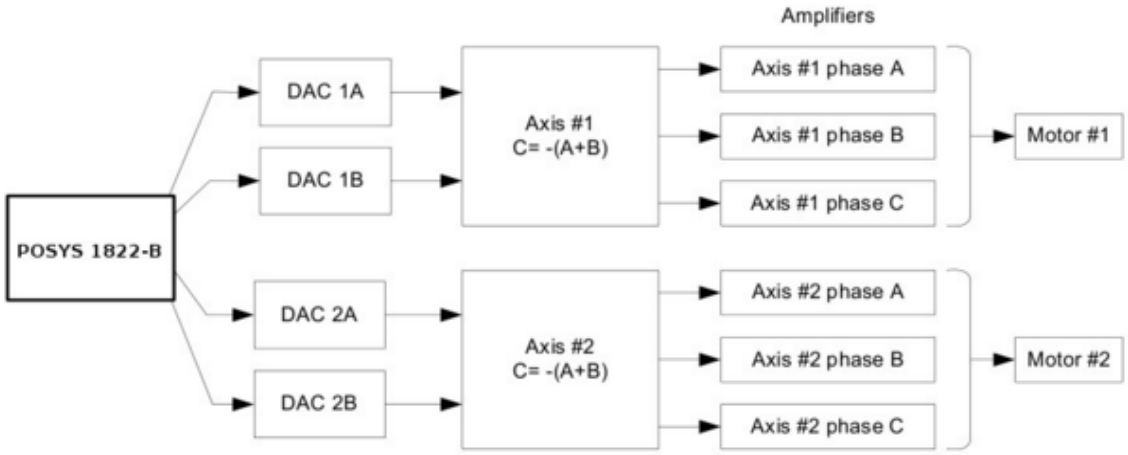

Abbildung 31: Typische Verstärkerkonfiguration für 3-Phasen Motor

Bei der DAC-Methode wird im Controller der digitale Wert mit Hilfe der DACs in einen analogen Wert konvertiert. Für jede Achse sind 2 DAC-Kanäle erforderlich. Die dritte Phase wird mit der Formel C = -(A+B) extern im Verstärker generiert. Dies wird mit Hilfe eines Operationsverstärkerschaltkreises On-board bewerkstelligt.

-o-

Hostkommunikation

Hostkommunikation

Die POSYS® 1800-B kommuniziert durch einer von bis zu drei möglichen Ports mit dem Host: entweder durch den bi-direktionalen Parallelport (ISA-Bus), einen asynchronen seriellen Port, oder via CAN 2.0B. Dies kann auch parallel geschehen.

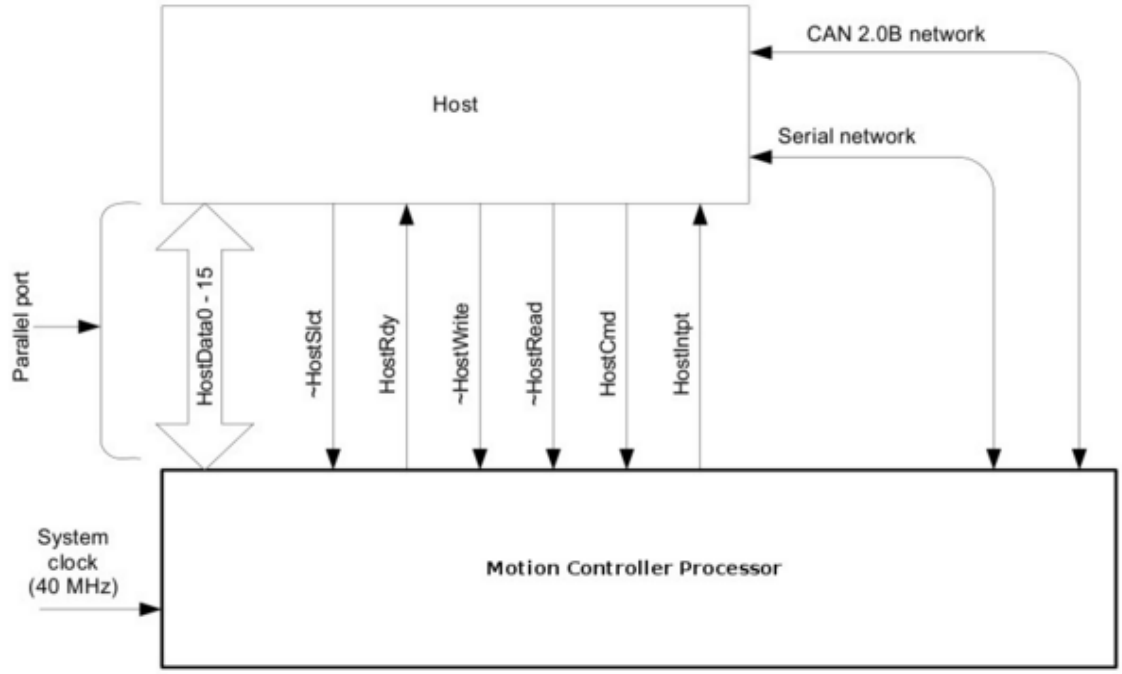

Abbildung 32: Host/Motion Controller Kommunikation

Die POSYS® akzeptiert Kommandos vom Host in einem Paketformat. Durch das Senden von Kommandos durch den Host kann das Verhalten der Positioniersteuerung entsprechend kontrolliert und der Status der POSYS® und der Motoren überwacht werden.

-o-

Host I/O Commands

Anmerkung: Einige Anwender ziehen es vor ihre eigenen Bibliotheken zu erstellen um mit der POSYS 1800-B zu kommunizieren; POSYS Motion Control's DLLs und Beispielprogramme für Delphi/C/C++ und VB, bieten aber auch dem Entwickler Softwaretools die ihm schnell zum Ziel führen eine anspruchsvolle und anwenderorientierte Arbeitsumgebung für die gewünschte Applikation zu erstellen.

-o-

Paralleler Kommunikationsport (ISA-Bus)

Bitte kontaktieren Sie POSYS Motion Control GmbH & Co.KG für Details.

-o-

Instruktionsfehler

Die POSYS überprüft die Kommandos die sie erhält auf Korrektheit. Dies soll die Sicherheit der ausgeführten Bewegung erhöhen indem ungültige Kommandowerte verworfen werden. Alle diese Überprüfungen die mit Host I/O Kommandos zu tun haben, werden als Instruktionsfehler bezeichnet. Um den Grund des Fehlers zu bestimmen, können Sie das Kommando GetHostIOError/GetInstructionError verwenden. Die Ausführung des Kommando's hat auch zur Folge, dass sowohl der Errorcode und der Errorbit im I/O Status Read Word gelöscht werden.

Die möglichen I/O Fehlercodes die der Motion Controller zurückliefern kann, sind in der folgenden Tabelle gelistet.

Code	Identifikation	Grund
0	No error	Kein Fehler.
1	Board reset	Prozessor Reset
2	Invalid instruction	Ungültiges Kommando
3	Invalid axis	Die Achsennummer in den oberen Bits des Anweisungswortes werden von diesem Controller nicht unterstützt.
4	Invalid parameter	Ungültiger Parameter
5	Trace running	Ein Kommando wurde ausgegeben welches ein gerade ablaufendes Tracing in seinem Ablauf ändern würde, während es läuft. Kommandos die einen solchen Fehler verursachen können sind SetTraceVariable, SetTraceMode & SetTracePeriod.
6	Reserved	
7	Block bound exceeded	1. Der mit dem Kommando SetBufferLength oder SetBufferStart gesendete Wert würde die physikalisch tatsächlich zur Verfügung stehenden Grenzen des Produktes überschreiten. 2. Entweder SetBufferReadIndex oder SetBufferWriteIndex haben einen Indexwert geschickt der größer oder gleich der Blocklänge war.
8	Trace zero	SetTraceStart Immediate wurde ausgeführt, aber die Länge des Tracebuffers ist noch auf “0” eingestellt.
9	Bad checksum	(gilt nur für die serielle Schnittstelle) Die Checksum die von der POSYS empfangen wurde, entspricht nicht der Checksum die vom Host geschickt wurde.
10	Reserved	
11	Negative velocity	Es wurde der Versuch gemacht eine negative Geschwindigkeit zu setzen, ohne dass sich die entsprechende Achse im Geschwindigkeitsmodus befindet.
12	S-curve change	Bei einer Achse die gerade ein S-Kurvenprofil ausführt, wurde der Versuch unternommen Profilparameter zu ändern. Dies ist nicht erlaubt.
13	Invalid move after event action	Nach einem Event Action das mit dem Kommando SetEventAction definiert wurde, wurde eine Bewegung versucht zu initiieren ohne das vorher das entsprechende Bit im Event Status Register gelöscht wurde.
14	Move into limit	Es wurde versucht eine Bewegung zu machen ohne das vorher die entsprechenden Endschalter Bit(s) im Event Status Register zurückgesetzt wurden.
15	Reserved	
16	Invalid operating mode change after event action	Es wurde der Versuch unternommen mit dem Kommando RestoreOperatingMode bzw. SetOperatingMode den Operating Mode des Controller's wieder herzustellen, ohne vorher im Event Status Register die Endschalterbits zu löschen.
17	Invalid instruction context	Kommando im gegenwärtigen Operating Mode, Systemstatus oder Kontext nicht erlaubt.

Tabelle 46: Instruktionsfehler

-o-

Serielle Schnittstelle

Konfiguration

Nach einem Reset liest die POSYS® einen 16 Bit Wert vom Peripheriebus (Adresse: 200h), welches verwendet wird um die Standardkonfiguration des seriellen Ports zu setzen.

Die serielle Schnittstelle kann auch mit dem Kommando SetSerialPortMode konfiguriert werden. Das POSYS 1800-B Programmier Handbuch beschreibt das Kommando näher. GetSerialPortMode liest die gegenwärtige Einstellung.

Das Konfigurationswort ist folgendermaßen aufgeschlüsselt um andere Konfigurationen zu laden:

Bit	Parameter	Indications	
0 – 3	Transmission rate selector	0 1 2 3 4 5 6 7	1,200 bits per second 2,400 bits per second 9,600 bits per second 19,200 bits per second 57,600 bits per second 115,200 bits per second 230,400 bits per second 460,800 bits per second
4-5	Parity selector	0 1 2 3	None Odd parity Even parity Reserved (do not use)
6	Number of stop bits	0 1	1 stop bit 2 stop bits
7-8	Protocol type	0 1 2 3	Point-to-point Multi-drop (idle line mode) Reserved (do not use) Reserved (do not use)
9-10	Reserved	–	
11-15	Multi-drop address selector Should be zero in point-to-point mode	0 1 ... 31	Address 0 Address 1 ... Address 31

Tabelle 47: Konfiguration der seriellen Schnittstelle

Kommunikations Modi	Standardwerte
RS-232	Standardeinstellung auf 57.6K, 1 Stopp Bit, no parity. Verwenden Sie SetSerialPortMode um den Wert zu ändern
RS-485	Anfänglich gesetzt auf 57.6K, 1 Stopp Bit, no parity, point-to-point mode. Danach gesetzt mit Kommando SetDefault.

Tabelle 48: Kommunikationsmodi

-o-

Kommandointerpretation

Die Kommandointerpretation die für die Kommunikation zwischen Host und Motion Controller verwendet wird, besteht aus einem Kommandopaket, welches vom Host gesendet wird, gefolgt von einem Antwortpaket, welches vom Motion Controller zurück gesendet wird. Der Host muss auf das Antwortpaket warten, es empfangen und dekodieren.

Kommandopakete vom Host enhalten folgende Felder.

Feld	Byte#	Beschreibung
Address	1	Ein Byte um zu identifizieren an welche POSYS das Kommandopaket geschickt wird. Dieses Feld sollte im Point-to-Point Mode immer Null sein.

Feld	Byte#	Beschreibung
Checksum	2	Ein 1 Byte Wert um Paketdaten zu verifizieren. Für Beschreibung siehe “Checksums.”
Instruction code	3-4	Eine 2 Byte Instruktion, das obere Byte wird zuerst gesendet (Achsennummer). Die Kommandostruktur ist dieselbe die im Parallelkommunikationsmodus verwendet wird.
Data	5	Null bis 6 Bytes an Daten, das meist signifikante Byte (MSB) zuerst. Die individuellen Kommandobeschreibungen sind im Detail im Programmierhandbuch beschrieben.

Tabelle 49: Format des gesendeten Kommandopaketes

Als Antwort sendet die POSYS® folgendes Datenpaket:

Feld	Byte#	Beschreibung
Address	(1)*	Ein Byte welches den Motion Controller identifiziert welches die Anwort schickt. Nur in Multi-Drop Mode.
Status	1 (2)	Null, falls das Kommando erfolgreich ausgeführt wurde, ansonsten wird ein Fehlercode der den Fehler näher spezifiziert, gesendet (siehe HostIO Errors).
Checksum	2 (3)	Ein 1 Byte Checksum Wert um die Paketintegrität zu verifizieren.
Data	3 (4)	Null bis 6 Bytes an Daten. Falls ein Fehler auftrat, werden keine Bytes gesendet (der Status-Byte is nicht Null). Falls kein Fehler auftrat, dann wird die Anzahl Bytes davon abhängen auf welches Kommando die POSYS® antworten sollte. Das meist signifikante Byte (MSB) wird immer zuerst gesendet.

Tabelle 50: Format des empfangenen Antwortpaketes

- Anmerkung: Der Adressbyte ist nur dann existent im Anwortpaket wenn der Kommunikationsmodus Mult-Drop ist. Es wird auch für die Checksum-Kalkulation herangezogen.

-o-

Instruktionsfehler

Der Motionprozessor vollführt eine Anzahl Überprüfungen abhängig vom Kommando, dass an die POSYS® gesendet wurde. Diese Überprüfungen erhöhen die Sicherheit des Positioniersystems indem offensichtlich inkorrekte Kommandodatenwerte eliminiert werden. All jene Überprüfungen die mit Host I/O Kommandos assoziiert sind, werden als Host I/O Errors bezeichnet. Der Status Byte im Antwortpaket kann eines der Fehlercodes enthalten wie im Kapitel „Instruktionsfehler“ beschrieben.

-o-

Checksummen

Beides, sowohl das Kommando als auch die Antwort enthalten ein Checksummen Byte. Die Checksumme wird verwendet um Übertragungsfehler zu ermitteln und erlaubt die POSYS® fehlerhafte Datenpakete zu identifizieren und abzulehnen.

Anders als beim Parallelport sind Checksummen bei seriellen Schnittstellen obligatorisch. Alle Kommandopakete die zur POSYS® geschickt wurden und die ungültige Checksummenwerte beinhalten, werden nicht weiter gegeben und ein Datenpaket mit einem Error Status Code wird zurückgeschickt.

Die serielle Checksumme wird kalkuliert indem alle Bytes addiert werden (mit Ausnahme des Checksummen Bytes), anschliessend wird das 2-er Kompliment des Ergebnisses ermittelt. Die acht niederwertigen Bits dieses Ergebnisses werden als Checksumme verwendet. Um eine korrekte Checksumme zu prüfen, sollten alle Bytes eines Pakets summiert werden (inkl. des Checksummen Bytes) und falls das Ergebnis der niederwertigen acht Bits = 0 ist, dann ist die Checksumme gültig.

-o-

Übertragungsprotokolle

Die POSYS® unterstützt die Möglichkeit zusätzliche Karten am seriellen Bus anzuhängen, ähnlich einer Kette, oder einem Netzwerk von POSYS®s die mit den gleichen seriellen Hardwaresignalen kommunizieren.

Drei Möglichkeiten bestehen mit der seriellen Schnittstelle Zeitprobleme, Übertragungsfehler und andere Vorgänge die während der seriellen Übertragung auftreten können, zu lösen. Da gibt es die Punkt-zu-Punkt Übertragung, falls nur eine Einheit verwendet wird, Multi-Drop Address-Bit Mode und Multi-Drop Idle-Line Mode, falls mehrere Karten an der seriellen Schnittstelle hängen. Die nächsten Sektionen beschreiben diese Übertragungsprotokolle.

-o-

Point-to-Point Mode

Dieser Modus ist für die direkte serielle Kommunikation zwischen POSYS® und Host vorgesehen, wenn eine direkte serielle Verbindung besteht zu einer Karte. In diesem Modus wird der Addressbyte nicht von der POSYS® verwendet (ausser zur Kalkulation der Checksumme) und die POSYS® antwortet auf alle Kommandos die vom Host kommen.

Wenn in diesem Modus, gibt es keine zeitkritischen Anforderungen an die Daten die übermittelt werden innerhalb eines Paketes. Die Menge an Daten die ein Kommandopaket beinhaltet, werden vom Kommando bestimmt im Datenpaket. Mit jedem Kommandopaket ist eine spezifische Menge an Daten verbunden/assoziiert. Wenn die POSYS® ein Kommandokode erhält wartet sie bis alle Daten empfangen wurden, ehe das Kommando ausgeführt wird. Auch die zurückgelieferte Menge an Daten die von einem Kommando angefordert wurde, ist bestimmt vom Kommandokode. Nach Ausführung eines Kommandos, wird die POSYS® ein Datenpaket in der erforderlichen Größe und Länge zurückliefern.

Wenn im Punkt-zu-Punkt Modus, gibt es für die POSYS® keine direkte Möglichkeit den Anfang eines neuen Datenpaketes zu unterscheiden, außer im Zusammenhang. Daher ist es für den Host wichtig in einem synchronisierten Zustand mit der POSYS® zu verbleiben wenn Daten gesendet und empfangen werden. Um sicher zu stellen die POSYS® verbleibt synchronisiert, wird empfohlen im Host eine Zeitspanne zu definieren auf die sie warten soll wenn Datenpakete von der POSYS® erwartet werden. Die empfohlene minimale Zeitspanne ist die Zeit die benötigt wird ein Byte bei der gewählten Baudrate plus 1 Millisekunde zu senden. z.B., bei einer Baudrate von 9600 Baud/sek. benötigt jedes Bit 1/9600 Sek. für die Übertragung. Ein typischer Byte besteht aus 8 Datenbits plus 1 Startbit und einem Stoppbit. Daher benötigt ein Byte gerade etwas mehr als 1 Millisekunde und die empfohlene Zeitspanne ist 2 Millisekunden. Falls es zu einer zeitlichen Überschneidung von zu empfangenden Datenbytes kommt während der Host auf Datenpakete wartet, dann sollte der Host davon ausgehen sich außerhalb der Synchronisation zu befinden. Um die Synchronisation wieder herzustellen, sollte der Host ein Byte ohne Daten schicken und auf die Antwort warten. Dieser Prozess sollte solange wiederholt werden, bis ein Datenpaket von der POSYS® empfangen wurde. An diesem Punkt werden Host und POSYS® wieder synchronisiert.

-o-

Multi-drop / Idle-line Mode

Von der POSYS® werden 2 verschiedene Multi-drop Protokolle unterstützt. Diese Modi werden vornehmlich dann auf einer seriellen Schnittstelle verwendet wenn ein einzelner Host mit mehreren POSYS® (oder anderen untergeordneten Geräten) kommunizieren soll. In diesem Modus wird zuerst ein Addressbyte benutzt, welches definiert für welches Gerät oder welche Einheit (POSYS®) das Datenpaket gedacht ist. Nur die adressierte Einheit wird auf das Gerät antworten bzw. reagieren. Daher ist es wichtig die Adressen der POSYS® sorgfältig auszuwählen (serielles Konfigurationswort, wie oben beschrieben) und das Addressbyte für jedes Kommandopaket einzubinden, dass für die POSYS® gedacht ist.

Da die Adresse, die einem Kommandopaket vorangeht, gedacht ist, die Antwort einer POSYS® im Multi-drop Mode zu aktivieren bzw. zu deaktivieren, muss das Multi-drop Protokoll eine Methode beinhalten die Synchronisation zw. dem Host und den verschiedenen POSYS® aufrecht zu erhalten, da es in einer solchen Umgebung schwierig ist erneut die Synchronisation herzustellen. Die beiden Multi-drop Protokolle differieren in der Methode wie sie die Synchronisation kontrollieren.

Bitte beachten Sie, dass die Multi-drop Protokolle auch in einer Punkt-zu-Punkt Konfiguration verwendet werden können, solange der Host immer das Addressbyte für jedes Kommandopaket mit vorne wegschickt und auch allen anderen Gesetzmäßigkeiten für das selektierte Protokoll folgt. Diese Methode der Handhabung erlaubt dem Host mit Sicherheit in einem synchronisierten Zustand zu verbleiben, ohne die Notwendigkeit irgendwelche Zeitspannen und

Resynchronisationsmethoden zu implementieren.

-o-

Idle-line Mode

Bei der Verwendung dieses Protokolls werden von der POSYS® strenge deterministische Auflagen an die Daten, die Teil eines Kommandos sind, gestellt. In diesem Modus interpretiert die POSYS® das 1. empfangene Byte nach einer Periode der Untätigkeit als den Beginn eines neuen Pakets. Alle bis dahin empfangenen Daten werden verworfen.

Die Timeout Periode ist gleich der Zeit die erforderlich ist 10 Bits an seriellen Daten mit der konfigurierten Baudrate zu senden (grob 1 Millisekunde bei 9600 Baud). Falls eine Verzögerung zwischen empfangenen Bytes eines Datenpakets in dieser Länge auftritt, werden die bereits empfangenen Bytes verworfen und das 1. Zeichen nach der Verzögerung als Addressbyte eines neuen Paketes gewertet.

-o-

Address Bit Protocol

Bei diesem Protokoll wird jedes übertragene Byte, dass von der POSYS® empfangen oder versendet wird, ein zusätzliches Bit enthalten (nach dem letzten Datenbit). Dieses Bit wird für die Identifikation des Addressbytes eines Paketes verwendet. Jedes Byte, dass von der POSYS® empfangen wird und bei dem dieses Bit gesetzt ist, wird als den Beginn eines neuen Kommandopakets interpretiert. Falls die POSYS® dabei war Daten eines früheren Kommandopakets zu empfangen, werden jene Daten verworfen.

-o-

Controller Area Network (CAN)

Controller Area Network (CAN)

CAN ist ein serielles Bus-System speziell um in Netzwerkumgebungen mit „intelligenten“ Gerätschaften als auch Sensoren und Aktoren innerhalb eines Systems oder Sub-Systems zu interagieren.

-o-

Überblick

Die POSYS® 1800-B können in ein CAN 2.0B Netzwerk integriert werden und mit anderen CAN Knoten koexisitieren aber nicht kommunizieren. Die POSYS® verwendet CAN um Kommandos zu empfangen, Antworten und optional auch asynchron Ereignisbenachrichtigungen zu senden. Jede Meldung hat folgende Adresse:

Message Type	CAN Address
Command received	0x600 + nodeID
Command response	0x580 + nodeID
Event notification	0x180 + nodeID

Tabelle 51: CAN Message Types

CAN Knoten kommunizieren mit Hilfe von Botschaften. Jede Botschaft kann bis zu 8 Bytes an Daten beinhalten. Der CAN Protokoll ist in der Lage automatisch Übertragungsfehler zu korrigieren, daher ist die Verwendung einer Checksumme nicht notwendig wie bei der seriellen Datenübertragung.

-o-

Message Format

CAN Knoten kommunizieren mit Hilfe von Botschaften und verwendet ein Standardformatidentifikatorlänge von 11 Bit. Alle Netzwerk Botschaften die das erweiterte Format mit 29 Bit verwenden werden ignoriert. Jede Botschaft kann bis zu 8 Bytes an Daten beinhalten. Der CAN Protokoll ist in der Lage automatisch Übertragungsfehler zu korrigieren, daher ist die Verwendung einer Checksumme nicht notwendig wie bei der seriellen Datenübertragung.

Die Datenformate der Meldungstypen werden in Termini von Bytesequenzen für die parallele Schnittstelle ausgedrückt. Kommandos haben unterschiedliche Längen; das POSYS® Programmierhandbuch zeigt dies auf.

Bytes, die immer vorhanden sind wenn ein Kommando ausgeführt wird, sind in der Spalte „Notwendig?“ entsprechend markiert. Die korrespondierenden Bytesequenzen für die 3 Meldungstypen sind wie folgt:

Command Received		
Message data byte	Required?	Corresponding parallel byte
1	Y	Command word, high byte
2	Y	Command word, low byte
3	N	1st data word, high byte
4	N	1st data word, low byte
5	N	2nd data word, high byte
6	N	2nd data word, low byte
7	N	3rd data word, high byte
8	N	3rd data word, low byte

Tabelle 52: CAN Kommando empfangen

Command Response		
Message data byte	Required?	Corresponding parallel byte
1	Y	Reserved (always zero)
2	Y	Instruction status code
3	N	1st data word, high byte
4	N	1st data word, low byte
5	N	2nd data word, high byte
6	N	2nd data word, low byte
7	N	3rd data word, high byte
8	N	3rd data word, low byte

Tabelle 53: CAN Kommando Antwort

Das erste Wort in der Antwort enthält den Wert Null (0) im oberen Byte und das untere Byte wird auch den Wert Null (0) enthalten, falls kein Fehler vorliegt, wird aber <> Null (0) falls ein Fehler auftrat während der Kommandoausführung (siehe Instruktionsfehler für weitere Informationen). Das Byte das dem Statusbyte folgt ist das obere Byte des ersten Datenbwortes, gefolgt vom unteren Byte des ersten Datenwortes usw. wie in der nachfolgenden Tabelle dargestellt.

Die eigentliche Anzahl zurückgelieferte Bytes wird bestimmt durch das ausgeführte Kommando. Im Programmierhandbuch sind die Kommandos mit den zurückgelieferten Werten gelistet.

Event Notification	
Message data byte	Data Interpretation
1	Reserviert (immer Null (0))
2	Achsennummer(0–3)
3	Event Status Register Wert, oberes Byte
4	Event Status Register Wert, unteres Byte

Tabelle 54: Event Benachrichtigung

Das erste Byte in einer Benachrichtigung enthält den Wert Null (0), das zweite Byte enthält den Wert für die Achse von dem die Nachricht stammt. Das dritte und vierte Byte sind das ober und untere Byte des Event Status Register's der benachrichtigenden Achse.

-o-

Konfiguration der CAN Schnittstelle

Nach einem Reset liest der Motion Controller einen 16 Bit Wert von seinem Peripheriebus welches dient die Standardkonfiguration für die CAN-Schnittstelle zu setzen.

Die CAN-schnittstelle des Motion Controller's kann mit dem Kommando SetCANMode an die eigenen Bedürfnisse angepasst werden. Diese Kommando wird benutzt um die CAN nodeID (0–127) als auch die Übertragungsrate eines speziellen Motion Controller's zu setzen. Die unterstützten Übertragungsraten sind wie folgt:

SetCANMode Encoding	CAN Übertragungsrate (bps)
0	1,000,000
1	800,000
2	500,000
3	250,000
4	125,000
5	50,000
6	20,000
7	10,000

Tabelle 55: Konfiguration der CAN-Schnittstelle

-o-

CAN Ereignis Benachrichtigung

Wenn die Kommunikation über die CAN-Schnittstelle erfolgt, kann die POSYS® optional Nachrichten senden, wenn im Event Status Register entsprechende Bits als aktiv gesetzt sind. Diese Möglichkeit steht in direktem Zusammenhang mit der Interrupt-Funktion des Motion Control Prozessors, wenn die parallele Schnittstelle (Board-intern) verwendet wird. Diese Meldungen werden mit der CAN-Adresse 0x180 + nodeID versendet.

Diese Funktionalität wird mit dem Kommando SetInterruptMask spezifiziert und kontrolliert. Für jedes Bit in der Benachrichtigungsmaske, dass als ON markiert ist, wird eine CAN-Nachricht generiert sobald das korrespondierende Bit im Event Status Register „1“ wird. Für weitergehende Informationen verweisen wir auf das Kapitel „Host Interrupts“.

-o-

EC Servomotorkontrolle

Überblick

Die POSYS® unterstützen zusätzlich zur Bahnprofilerstellung und geschlossenem Servoregelkreis auch die Möglichkeit das Kommutierungssignal für 2- und 3-phasige bürstenlose Servomotoren zu generieren (Modell-abhängig). Dies erlaubt die gemischte Benutzung von bürstenlosen und bürstenbehafteten Servomotoren mit ein und dem selben Controller.

Der Kommutierungsteil des Controllers verwendet einen Eingang des Motorausgangssignals, entweder vom Servofilter oder vom Motorkommandoregister (abhängig ob die Achse im Closed Loop oder Open Loop Mode betrieben wird). Dieses vorkommutierte Kommandosignal wird mit Kommutierungswerten, hergeleitet von einer internen Sin/Cos Tabelle, multipliziert.

Der Kommutierungswinkel in der Sin/Cos Tabelle wird sowohl von der Enkoderposition als auch von Parametern die vom Hostprozessor stammen, bestimmt, die den spezifischen Enkoder ins Verhältnis zu den magnetischen Polen des Motors setzen.

Zwei Wellenformen für die Kommutierung werden unterstützt, eine für 3-phasige Motoren mit 120 Grad Winkel zwischen den Phasen (z.B. bürstenlose Servomotoren) und eine für 2-phasige Motoren mit 90 Grad zwischen den Phasen (z.B. Schrittmotoren).

Andere Besonderheiten der POSYS® sind die Fähigkeit Hallsensoreingänge für die Phaseninitialisierung zu verwenden, einen Indextakt um die Kommutierungssynchronisation zu erhalten, den Enkodereingang vor zu skalieren um eine größere Anzahl verschiedene Arten von Rückmeldeeinheiten zu unterstützen und die Unterstützung von geschwindigkeits-basierter Phasenvorschau für weichere und effizientere Hochgeschwindigkeitsoperation.

-o-

Phasenanzahl

Die POSYS® unterstützt 2 verschiedene Signalformen für die Kommutierung. Die Eine ist eine um 120° versetzte Signalform, geeignet für 3-Phasen bürstenlose Servomotoren, die andere eine um 90° versetzte Signalform, geeignet für 2-Phasen bürstenlose Motoren. Um die eine oder andere Signalform zu aktivieren, wird das Kommando SetMotorType verwendet. Eingestellte Werte werden mit dem Kommando GetMotorType gelesen.

Abhängig von der Signalform und des gewünschten Motorausgangssignals (PWM oder DAC), werden entweder 2 oder 3 kommutierte Ausgabesignale vom Controller pro Achse ausgegeben. Die folgende Tabelle listet die verschiedenen Kombinationen auf:

Signalform	Motorausgabemodus	Anzahl Ausgabesignale & Name
3 Phasen	PWM5050	3 (A, B, C)
3 Phasen	PWMSign/Mag	2 (A, B)
3 Phasen	DAC	2 (A, B)
2 Phasen	PWM5050	2 (A, B)
2 Phasen	PWMSign/Mag	2 (A, B)
2 Phasen	DAC	2 (A, B)

Tabelle 56: Signale je nach Motorausgabemodus

Das nachfolgende Diagramm zeigt die Phasen A, B und C für einen 3-Phasen bürstenlosen Motor und die A- und B-Phasen für einen 2-Phasen bürstenlosen Motor.

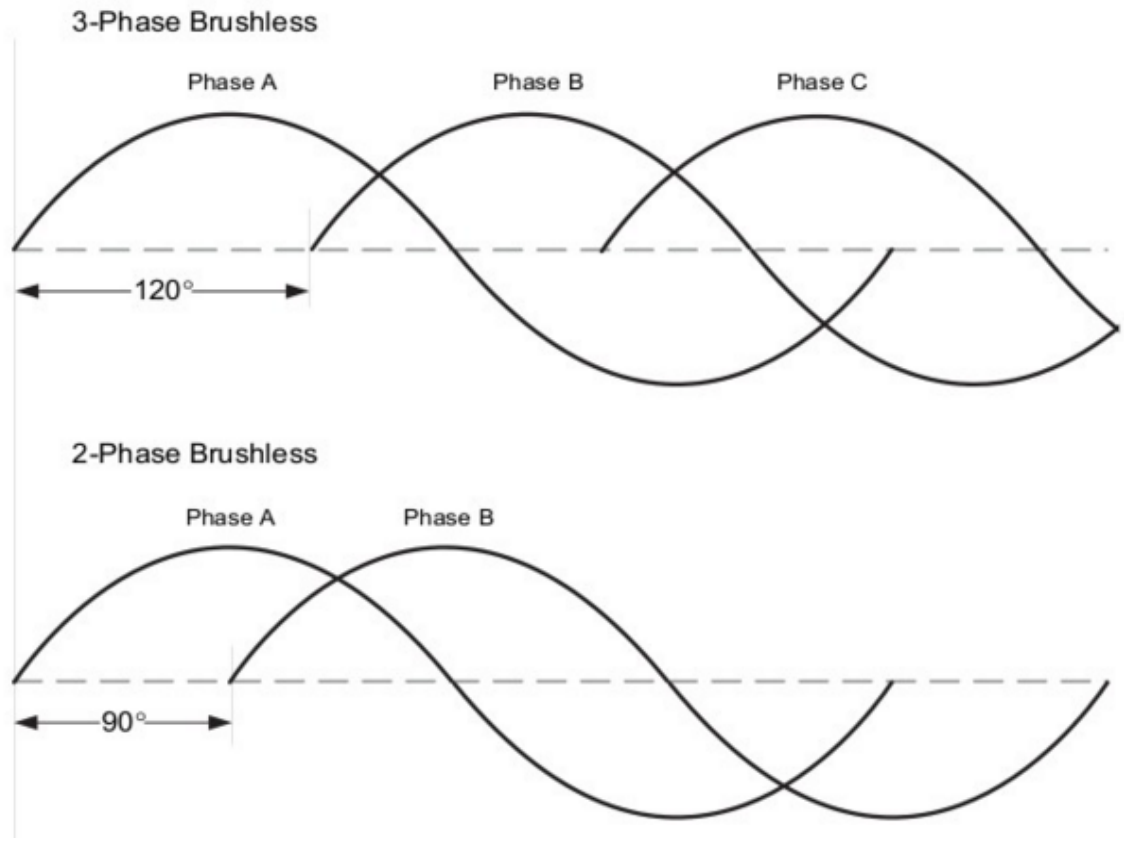

Abbildung 33: Signalformen für bürstenlose DC Servomotoren

-o-

Phasenkontrollmodule

Um einen bürstenlosen DC Servomotor direkt anzusteuern, ist es notwendig den Winkel des Rotor's des Motor's zu kennen, da es sich ständig ändert. Dies kann auf zweierlei Arten geschehen. Die erste Methode ist durch Verwendung der Hallsensoren, die zweite durch Verwendung der Positionsrückmeldung durch einen Enkoder. In beiden Fällen ist es notwendig, dass die Sensoren direkt mit der Welle verbunden sind. Vorteilhafter ist die Verwendung von Enkodern, da diese eine weichere Bewegung und allübergreifend bessere Leistung erzielen als Hallsensoren.

Das SetCommutationMode wird verwendet, um die Kommutierungsmethode (Enkoder- oder Hallsensorbasiert) auszuwählen. Welcher Wert eingestellt ist kann mit dem Kommando GetCommutationMode überprüft werden.

Oft werden sowohl Enkoder als auch Hallsensoren verwendet. Die Hallsensoren dienen der Phaseninitialisierung, während anschließend die Enkoder verwendet werden um die korrekte Wellenform für die Phasenausgabe zu generieren während der normalen Motorbetriebes.

-o-

Phasenzählungen

Für die Sinuskommutierung ist es notwendig die Anzahl Enkoderstriche pro elektrischen Zyklus zu definieren. Um diese zu bestimmen müssen sowohl die Enkoderauflösung pro Motorumdrehung als auch der Wert für die Anzahl elektrischer Zyklen des Motors bekannt sein. Wenn diese beiden Werte bekannt sind, kann die Anzahl Enkoderauflösungen pro elektrischen Zyklus mit folgender Gleichung ermittelt werden:

Counts per cycle = Counts_per_rot/electrical cycles

dabei ist:

Counts_per_rot ist die Enkoderauflösung pro Motorumdrehung
electrical cycles ist die Anzahl elektrischer Zyklen

Für die Anzahl elektrischer Zyklen verweisen wir hier auf die Spezifikationen des Motorenherstellers. Normalerweise ist die Anzahl elektrischer Zyklen identisch mit der Hälfte der Anzahl der Pole. Bitte verwechseln Sie nicht die Anzahl Pole mit der Anzahl Polpaare. z.B. falls die Dokumentation eines Motors ausweist, dass die Enkoderauflösung 1024 ist und 4 Pole besitzt, dann sieht die Berechnung wie folgt aus:

Counts per cycle = 1,024 Zählungen_pro_Umdr./2 elektrische Zyklen pro Umdrehung
Counts per cycle = 512

Das Kommando um die Anzahl Enkoderauflösung pro elektrischen Zyklus zu setzen ist SetPhaseCounts. Das Kommando GetPhaseCounts liest den programmierten Wert zurück.

-o-

Phaseninitialisierung

Nach dem Einschalten des Systems muss der Controller den korrekten Kommutierungswinkel relativ zur Enkoderposition bestimmen. Diese Information wird durch eine Prozedur, genannt Phaseninitialisierung, bestimmt und ermittelt.

Der Controller unterstützt 3 Methoden der Phaseninitialisierung; algorithmisch, Hall Sensor-basiert, und Direct-set.

-o-

Hallsensor-basierte Phaseninitialisierung

Um den Controller für die Hallsensor-basierte Initialisierung zu verwenden, wird das Kommando SetPhaseInitializeMode und der Parameter Hall-basiert verwendet.

In diesem Modus werden 3 Hallsensorsignale verwendet um die Phasen des Motors zu bestimmen und die Sinuskommutierung beginnt automatisch nachdem der Motor eine volle Umdrehung absolviert hat.

Die Anschlüsse für die Hallsensoren werden über den optionalen Stecker mit den Kennzeichnungen Hall1A-C (Achse #1) und Hall2A-C (Achse #2) usw. zurückgeführt. Es sollte Sorgfalt angewandt werden beim Anschliessen der Hallsensoren. Um den gegenwärtigen Status der Hallsensoren zu erfahren, kann das Kommando GetSignalStatus verwendet werden.

Abbildung 34 illustriert das Verhältnis zwischen dem Status der drei Hallsensoreingängen, der sinus-kommutierten Wicklungserregung der jeweiligen Phase und Phase-zu-Phase BEMF Signalformen während einer Vorwärtsbewegung. Der Motion Controller erwartet eine 120° Aufteilung zwischen den Hallsignaländerungen. Um Hallsensoren mit 60° Abstand zu kommutieren tauschen und invertieren Sie die entsprechenden Hallsignale und Motorphasen um die zu erwartenden Zustände der Hallsensoren zu generieren. Dieses „Hall zu BEMF Phasendiagramm“ ist der gebräuchlichste Weg die entsprechende Lage zu spezifizieren und ein ähnliches Diagramm wird normalerweise vom Motorhersteller zur Verfügung gestellt.

Mit der Hall-basierten Phaseninitialisierung sind keine weiteren Prozeduren für die Motorenkonfiguration erforderlich. Die Initialisierung erfolgt mit der Ausführung des Kommando's InitializePhase und greift unverzüglich ohne weitere Motorbewegung.

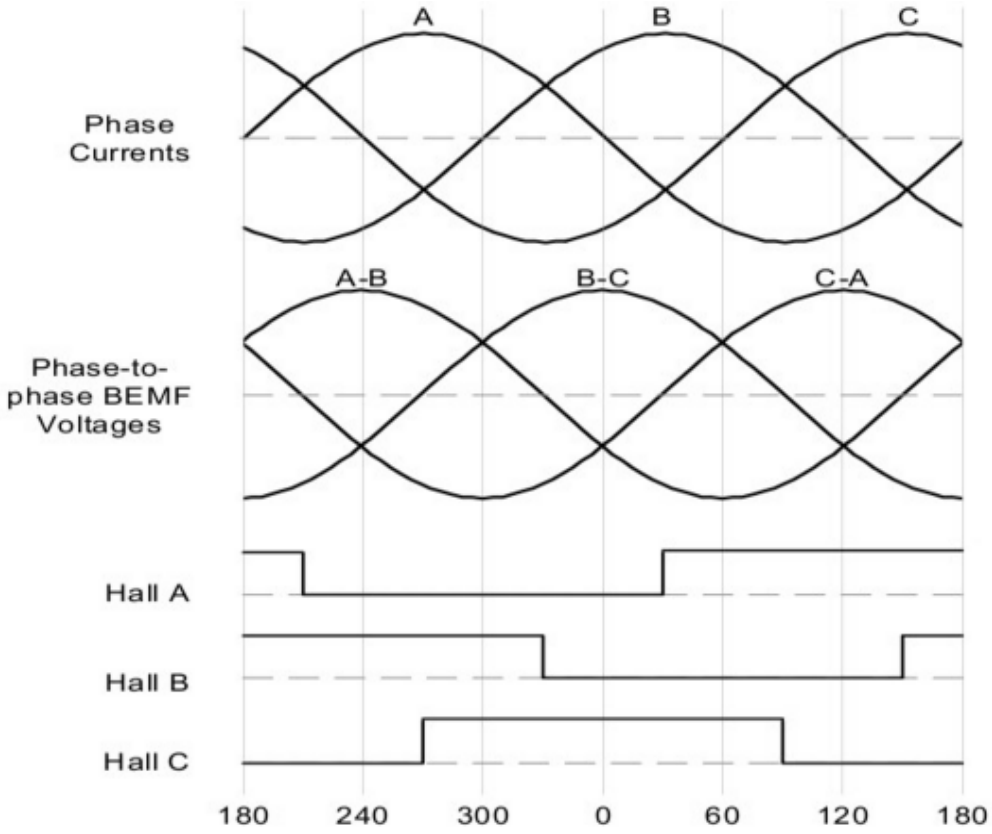

Abbildung 34: Hall-basierte Phaseninitialisierung

Um sich den verschiedenen Typen von Hallsensoren anzupassen, auch jenen mit invertierter Schaltung, kann die Signalinterpretation der Signallogik vom Host gesetzt werden. Das Kommando SetSignalSense akzeptiert ein Bit-programmiertes Wort das überprüft ob die ankommenden Hall-Signale als aktiv "high" oder aktiv "low" interpretiert werden. Um den programmierten Hall-Interpretationswert zurückzulesen wird das Kommando GetSignalSense verwendet.

-o-

Algorithmische Phaseninitialisierung

Das Kommando SetPhaseInitializeMode mit dem Parameter algorithmic versetzt den Controller in diesen Modus.

Für diesen Modus benötigt der Motor keine anderen Informationen als die die von den Rückmeldeeinheiten zurückgeliefert werden. Um die Phasen zu bestimmen, stimuliert der Controller kurz die Wicklungen des Motors und setzt die Phaseninitialisierung aufgrund der Motorreaktion. Aufgrund der resultierenden Bewegung kann der Controller automatisch die Phasen korrekt bestimmen.

Abhängig von der Größe und der Geschwindigkeit des Motors, kann die Phaseninitialisierung unterschiedlich lange dauern. Um diese Unterschiede anzupassen, kann die Wartezeit bis der Motor zum Stillstand kommt, mit dem Kommando SetPhaseInitializeTime programmiert werden. Das Kommando GetPhaseInitializeTime liest den programmierten Wert zurück.

Um die Wirkung auf die mechanischen Systemkomponenten so gering wie möglich zu halten, verwendet dieses Methode einen Drehmomentsollwert, der vom Host gesetzt wird, um die Maximalmenge an Strom für die Phaseninitialisierung zu bestimmen. Als typische Werte sollte man von einem Wert der zwischen 5% und 25% des maximal benötigten Ausgangstromes liegt ausgehen, aber in der Regel 3-mal höher als der Anfahrwiderstand ist. Um bestmögliche Resultate zu erzielen, kann der Wert auch experimentell ermittelt werden. Das Kommando für den Motorbefehlswert ist SetMotorCommand. Der programmierte Wert kann mit dem Kommando GetMotorCommand zurückgelesen werden.

Um die Initialisierungsprozedur auszulösen, verwendet man das Kommando InitializePhase. Dieses Kommando wird sofort ausgeführt, ein Update-Kommando ist nicht notwendig.

Jedoch, ehe das Kommando für die Phaseninitializierung ausgeführt wird (InitializePhase), muss der Motor ausgeschaltet (SetMotorMode), ein Drehmomentsollwert definiert (SetMotorCommand) und die Zeitspanne für die Initialisierung bestimmt sein (SetPhaseInitializeTime).

Während der Phaseninitialisierung nach der Algorithmusmethode kann sich der Motor plötzlich in die eine oder andere Richtung bewegen. Es müssen Vorsichtsmaßnahmen ergriffen werden um Schäden zu vermeiden. Zusätzlich, um akkurate Ergebnisse zu erzielen, muss die Bewegung in beide Richtungen ungehindert ablaufen können und darf nicht durch übermäßige Anfahrreibung behindert werden.

-o-

Direct-Set Phaseninitialisierung

Falls nach dem Einschalten die Phasen bekannt sind, können diese explizit mit dem Kommando SetPhaseAngle gesetzt werden.

Dies kommt typischerweise dann vor, wenn Sensoren wie Resolver benutzt werden, bei der zurückgelieferten Motorpositionsinformationen absolut sind (nicht inkremental), und benutzt werden können um ein Rechteckdatensignal zu generieren als auch vom Hostrechner direkt gelesen werden können.

-o-

Programmierung der Phaseninitialisierung

Die folgenden Beispiele zeigen typische Kommandofolgen eines bürstenlosen Motors für alle vier Initialisierungsmethoden, um die Kommutierung zu initialisieren.

-o-

Hallsensor-basierte Initializierungssequenz

SetMotorType xx	Motortyp setzen
SetOutputMode m	Motorausgabemodus setzen
SetPhaseCounts uuuu	Encoderauflösung pro elektrischen Zyklus setzen
SetSignalSense vvvv	Interpretation der Signalerkennung auf Hallsensorerkennung setzen
SetPhaseInitializeMode Hall	Phaseninitialisierungsmethode auf Hallsensor-basiert setzen
InitializePhase	Initialisierung starten

Tabelle 57: Hall-basierte Initialisierungssequenz

Diese Sequenz setzt den Controller in den Modus die Hallsensor-basierte Initialisierung anzuwenden und die Phaseninitialisierung sofort zu beginnen. Der Motor wird sich nicht bewegen aufgrund dieser Sequenz und es gibt keine Verzögerung bei der Ausführung weiterer Operationen.

-o-

Algorithmische Initialisierungssequenz

SetMotorType xx	Motortyp setzen
SetOutputMode m	Motorausgabemodus setzen
SetPhaseCounts uuuu	Anzahl der Encoderstriche pro elektrischen Zyklus setzen
SetPhaseInitializeMode Algorithmic	Algorithmische Initialisierung setzen
SetOperatingMode zz	Setze Operating Modus auf “trajectory and position loop disabled”
SetPhaseInitializeTime wwww	Initialisierungsdauer der Phaseninitialisierung setzen.

SetMotorCommand yyyy	Einstellen des Initialisierungsmotorausgabesignals.
InitializePhase	Initialisierung starten.

Tabelle 58: Algorithmische Initialisierungssequenz

Diese Sequenz veranlasst den Motor sofort mit der Initialisierung zu beginnen die "wwww" Anzahl von Servoloops andauert. Um zu bestimmen ob die Prozedur komplettiert wurde, kann man mit dem Kommando GetActivityStatus überprüft werden. Der "Phase Initialization" Bit wird anzeigen ob die Prozedur beendet wurde. Nach dem der Initialisierungsprozess beendet wurde, sollte der Motor mit dem Kommando SetOperatingMode eingeschaltet werden, falls der Motor im Closed-Loop-Mode betrieben werden soll.

-o-

Direct-Set Initialisierungssequenz

SetMotorType xx	Motortyp setzen
SetOutputMode m	Motorausgabemodus setzen
SetPhaseCounts xxxx	Anzahl der Encoderstriche pro elektrischen Zyklus setzen
SetPhaseAngle yyyy	Setzen des Phasenwinkels basierend auf Informationen vom externen Sensor.

Tabelle 59: Direct-Set Initialisierungssequenz

Diese Sequenz wird den Phasenwinkel direkt setzen, der von einem externen Sensor bestimmt wurde. Der Wert muss zwischen 0 und der maximalen Enkoderauflösung pro elektrischer Zyklus sein.

-o-

Index-Taktreferenzierung

Um die Langzeitverlässlichkeit der Kommutierung zu verbessern, unterstützt die POSYS® die Möglichkeit einen Indextakt des Enkoders als Referenzpunkt während der Kommutierung zu verwenden. Durch Verwendung des Indextaktes während der Phasenkalkulation wird jedweder Langzeitverlust an Enkoderzählungen, welche die Genauigkeit der Kommutierung beeinflussen können, eliminiert.

Um die Referenzierung des Indextaktes verwenden zu können, muss dieses Signal vom gewählten Enkodertyp zur Verfügung gestellt werden und zwar einmal pro Umdrehung.

Die Referenzierung des Indextaktes wird für alle bürstenlosen Motoren mit Inkrementalenkoder empfohlen. Für bürstenlose Linearmotoren wird er normalerweise nicht benötigt, obwohl er verwendet werden kann, solange der Indextakt so angelegt ist, dass jeder Takt im gleichen Phasenwinkel innerhalb des Kommutierungszyklus auftritt.

Die Verwendung des Indextaktsignals setzt nicht voraus, dass der Wert für die Enkoderauflösung pro elektrischen Zyklus ein Integer ist. Für den Fall es ist kein Integer sollte auf- oder abgerundet werden.

Umgekehrt, wenn kein Indextakt verwendet wird, dann muss die Enkoderauflösung pro elektrischen Zyklus exakt ein Integer ohne Rest sein.

z.B. bei Verwendung eines bürstenlosen Motors mit 6 Polen wäre ein Enkoder ohne Indextakt und 1200 Inkremente pro Umdrehung eine gute Wahl, aber ein Enkoder mit 1024 Inkremente wäre dies nicht, da 1024 nicht teilbar durch 3 ist ohne Rest.

Mit dem Kommando Set/GetPhaseCorrectionMode kann die Indextaktphasenkorrektur aktiviert bzw. deaktiviert werden.

Die Indextaktreferenzierung wird automatisch vom Controller ausgeführt, egal welcher Initializationsschema gewählt wurde (algorithmic, Hall-based, microstepping oder Direct Set).

-o-

Phasenfehlererkennung

Mit einem korrekt installierten Indextakt wird der Controller automatisch jedweden Verlust an Enkoderzählungen ausgleichen und korrigieren.

Jedoch sollte der Verlust größer werden, oder der Indextakt kommt nicht an innerhalb der erwarteten Position während des Kommutierungszyklusses, dann spricht man von einem „Kommutierungsfehler“. Der Bit 11 im Event Status Register für Kommutierungsfehler wird gesetzt.

Dieses Bit wird immer dann gesetzt falls die erforderliche Korrektur größer als (PhaseCounts/128)+4 ist.

Kommutierungsfehler können durch eine Vielzahl Umstände verursacht werden. Die Häufigsten sind im Folgenden gelistet:

- elektrische Störungen auf den A und B Enkoderleitungen
- elektrische Störungen auf der Leitung für den Indextakt
- falsches Setzen der Enkoderauflösung pro elektrischen Zyklus

Für jede Instanz in der ein Kommutierungsfehler auftritt wird die Phasenreferenzierung für den Indextakt ausgesetzt.

Abhängig vom Grund des Kommutierungsfehlers, könnte es ein einmaliges Ereignis sein oder kontinuierlich auftreten.

Sobald ein Kommutierungsfehler auftritt wird Bit#11 im Event Status Register „high“ gesetzt (1). Diese Bedingung kann auch als Quelle für die Verwendung von Hostinterrupts verwendet werden, um den Host automatisch über den aufgetretenen Kommutierungsfehler zu unterrichten. Dieses Bit muss vom Host zurückgesetzt werden, jedoch muss bedacht werden, dass je nach der Art des Fehlers der Kommutierungsfehler kontinuierlich auftreten kann.

Ein Kommutierungsfehler kann anzeigen, dass mit der Konfiguration des Positioniersystems ein ernsthaftes Problem besteht, welches sich in unsicheren Bewegungsbläufen äußert. Es liegt in der Verantwortung des Hosts/Bedieners den Fehler zu bestimmen und zu korrigieren um sichere Bewegungsabläufe zu gewährleisten.

-o-

Justierung der Phasenwinkel

Die POSYS® für bürstenlose Servomotoren verfügen über die Fähigkeit den Kommutierungswinkel des Motor's direkt zu ändern, im Stillstand als auch während der Bewegung. Obwohl dies nicht unbedingt notwendig ist, kann es zum Testen ganz nützlich sein, oder während der Kommutierungsinitialisierung im Mikroschritt bzw. Direkt-Set Modus.

Um den Kommutierungswinkel zu ändern, wenn der Motor stillsteht, wird das Kommando SetPhaseAngle verwendet.

Um den Kommutierungswinkel zu ändern, während sich der Motor bewegt, ist auch der Indextakt notwendig und das Kommando SetPhaseOffset wird verwendet, welches nur dann ausgeführt wird, wenn ein Indextakt erfasst wird. Die folgende Beschreibung liefert einiges Hintergrundwissen zu dieser Funktion.

Nach der Phaseninitialisierung wird der korrekte Kommutierungswinkel vom Controller als Relativwert vom Index (in Inkrementen) zum maximalen Ausgabewert der Phase A gespeichert (commutation "zero" location). Dieser 16-Bit Registerwert kann mit dem Kommando GetPhaseOffset gelesen werden.

Die folgende Grafik zeigt die Abhängigkeit zwischen der Kommutierungsnullposition der Phase A, der Indexposition und dem Offsetwert der Phase. Der gezeigte Indextakt für einen gegebenen Motor in dieser Grafik könnte irgendwo innerhalb eines Phasenzyklusses lokalisiert worden sein, da es von Motor zu Motor variieren kann. Nur bei Motoren die dahingehend montiert wurden, dass die Indexposition immer im gleichen Bezug zu den Motorwicklungen stehen, werden eine konsistente Indexposition relativ zur Kommutierungsnullposition aufweisen.

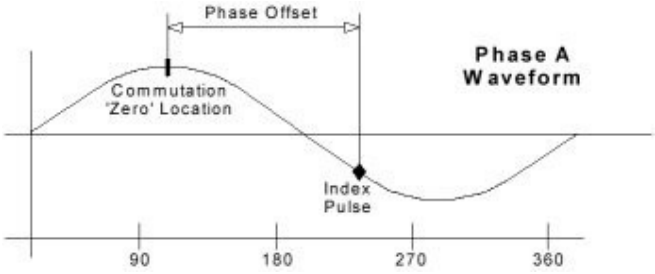

Abbildung 35: Justierung des Phasenwinkels

Der Phasenoffsetregister wird vor der Phaseninitialisierung den Wert ffff (hex) aufweisen. Nach der Phaseninitialisierung und nachdem der Motor mindestens eine Umdrehung dahingehend vollführt hat, dass ein Indextakt erfasst werden konnte, wird der Phasenoffsetwert als positive Nummer mit einem Wert zwischen 0 und der Enkoderauflösung pro elektrischen Zyklus gespeichert.

Anmerkung: Falls eine Achse im Dual Encoder Loop Mode mit einer Auxiliary Achse verknüpft ist, müssen die Kommandos SetPhaseAngle, SetPhaseOffset und SetPhaseCounts mit der Hauptachse abgeglichen werden.

Um den Phasenoffsetwert von einer Nummer in Grad zu konvertieren, kann folgender Ausdruck verwendet werden:

Offsetdegrees = 360 * Offsetcounts/counts_per_cycle

wobei

Offsetdegrees ist der Phasenoffset in Grad
Offsetcounts ist der Phasenoffset in Enkoderstriche
counts_per_cycle ist die Anzahl Enkoderstriche pro elekctrischen Zyklus die mit dem Kommando SetPhaseCounts gesetzt wurde.

Der Phasenoffsetwert kann, während der Motor in Bewegung ist, beliebig oft verändert werden, wobei nur kleine Änderungen vorgenommen werden sollten um plötzliche, große Sprünge zu vermeiden.

Die Kommandos SetPhaseOffset und GetPhaseOffset können nur verwendet werden, wenn ein Enkoder mit Indextakt angeschlossen ist. Wenn der Indextakt nicht angeschlossen oder verwendet wird, kann der Phasenoffsetwinkel nicht justiert bzw. zurückgelesen werden.

Die relative Phasenpositionierung von B und C zu A werden durch das Setzen von Phasenoffsetwerten nicht verändert. Diese Phasen bleiben bei 90° bzw. 120° Offset zu Phase A (abhängig von der gewählten Signalform).

-o-

Encoder Prescaler

Besonders in Verbindung mit Linearmotoren kann die Enkoderauflösung pro elektrischen Zyklus stark variieren. Typischerweise haben Motoren Enkoderzählwerte zwischen 1 und 32767. Lineare bürstenlose Motoren dagegen können Werte von 1.000.000 pro Zyklus und höher haben wegen der öfteren Verwendung von laser-basierten Enkoder mit höchster Genauigkeit.

Um diesen großen Bereich abzudecken, verfügen die POSYS® 182x-B für bürstenlose Servomotoren über eine Preskalarfunktion welches, für den Zweck der Kommutierungskalkulationen, die eingehenden Inkremente durch 64, 128 oder 256 teilt. Mit der aktivierten Preskalarfunktion kann der maximale Bereich von Inkrementen pro elektrischer Zyklus dann 8.388.352 betragen.

Das Kommando SetPhasePrescale On aktiviert den Modus. Das Kommando SetPhasePrescale Off deaktiviert ihn.

Die Preskalarfunktion hat nur auf die Kommutierung des Controller's Einfluss. Es beeinflusst nicht die Position durch die Servoregelung oder abgefragt durch das Kommando GetActualPosition.

HINWEIS Die Preskalarfunktion sollte weder aktiviert noch deaktiviert werden wenn der Motor sich in Bewegung befindet.

-o-

Sinuskommutierung

Abbildung 36 bietet einen Überblick über die Flusskontrolle der Sinuskommutierung einer Achse der POSYS® 182x-B wenn sie im bürstenlosen Modus betrieben wird. Um die Sinuskommutierung auszuwählen, sollte mit dem Kommando SetCommutationMode Enkoder-basiertes Phasing ausgewählt werden.

Im Vergleich zur Hallsensor-basierten Kommutierung bietet die Enkoder-basierte Sinuskommutierung verbessertes, vor allem weicheres Laufverhalten der Motoren und bessere Positioniergenauigkeit aufgrund fehlender kommutierungs-bedingter Drehmomentunterbrechungen.

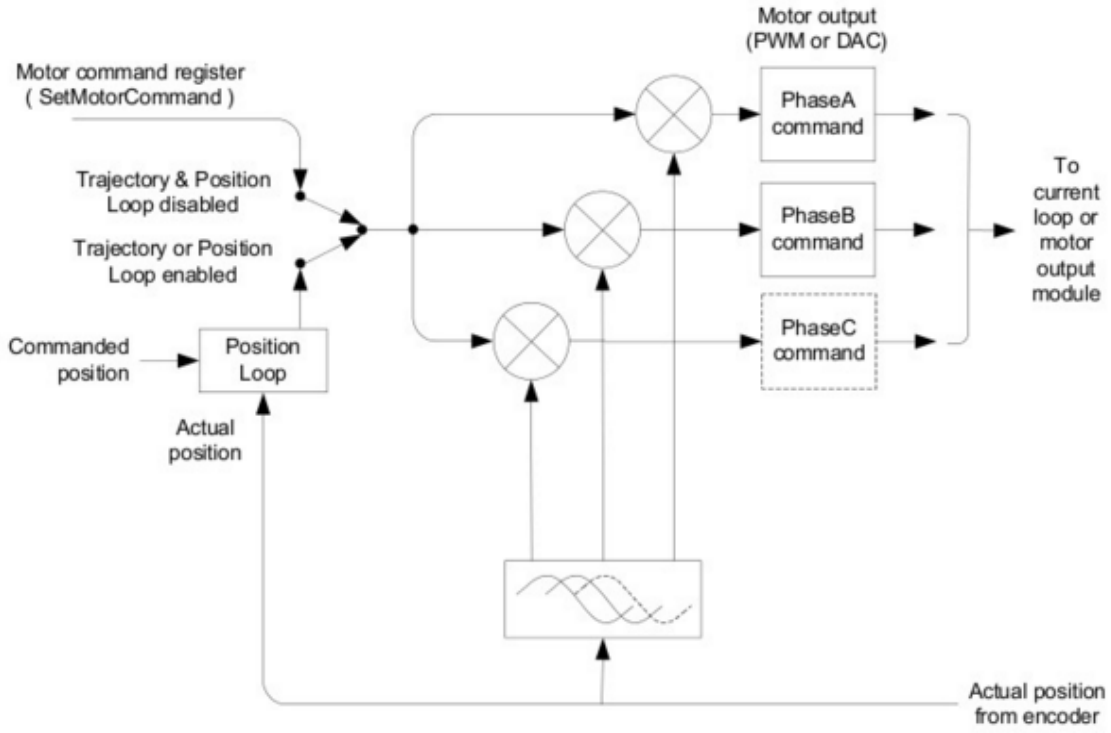

Abbildung 36: Sinuskommutierung

Die Logik der Sinuskommutierung des Motion Controller's verwendet als Referenz das Motor Command Signal entweder vom Position Servo Loop oder vom Motor Command Register (abhängig davon ob das Position Loop Modul aktiviert ist oder nicht). Dieses vorkommutierte Kommandosignal wird mit Kommutierungswerten, hergeleitet von einer internen Sin/Cos Tabelle, multipliziert.

Der Kommutierungswinkel in der Sin/Cos Tabelle wird sowohl von der Enkoderposition als auch von Parametern die vom Hostprozessor stammen, bestimmt, die den spezifischen Enkoder ins Verhältnis zu den magnetischen Polen des Motors setzen.

Zwei Wellenformen für die Kommutierung werden unterstützt, eine für 3-phasige Motoren mit 120 Grad Winkel zwischen den Phasen (z.B. bürstenlose Servomotoren) und eine für 2-phasige Motoren mit 90 Grad zwischen den Phasen (z.B. Schrittmotoren).

Sobald die Kommutierung abgeschlossen wurde, werden die einzelnen Motorausgangssignale für die einzlnen Phasen (A, B, C wie bei einem 3-Phasen bürstenlosen Servomotor) direkt auf den Verstärker oder auf das Current Control Modul ausgegeben. Falls direkt zum Motor werden die Signale in eine der möglichen Ausgabeformate (PWM oder DAC) konvertiert.

Um die indivuellen Phasenkommandos zu lesen, verwenden Sie das Kommando GetPhaseCommand.

-o-

Schrittmotorkontrolle

Überblick

Die POSYS 1800-B Motion Controller unterstützen eine Anzahl spezieller Funktionen für Schrittmotoren. Dies sind Generierung von Takt- & Richtungssignale, Unterstützung für direkte Ansteuerung von Mikroschrittmotoren und eine Funktion für das Haltemoment. Generell gesagt werden zwie Arten von Schrittmotoren unterstützt, Takt- & Richtungssignalmotoren und Mikroschrittmotoren. Takt & Richtungssignalmotoren sind Schrittmotoren die von einem Verstärker angetrieben werden, welches Takt- & Richtungssignale akzeptiert. Mikroschrittmotoren hingegen sind Motoren die mit einem Verstärker verbunden werden, welcher die einzelnen Phasen direkt ansteuert ähnlich wie bei bürstenlosen Servomotoren.

Im Großen und Ganzen sind die Funktionen der POSYS wenn sie mit Schrittmotoren verwendet wird, ähnlich oder gleich denen wie bei der Verwendung mit Servomotoren. Die Bahngenerierung, Breakpoints, Tracefunktionen und eine Anzahl weiterer Funktionen sind absolut identisch und Motor-unabhängig. Der Hauptunterschied zwischen Servo- und Schrittmotoren jedoch besteht darin dass das Position Loop Modul für Schrittmotoren nicht verwendet werden kann und die Art der Motorausgabesignalerstellung in einigen Aspekten differiert im Vergleich zu denen für Servomotoren.

Es gibt auch eine Anzahl Funktionen die vom Prinzip ähnlich sind aber von der Implementierung abweichen. Der Schleppabstandsfehler wie für Servomotoren verwendet, ist das Äquivalent zur Stall Detection für Schrittmotoren.

Die Unterschiede werden in den nächsten Kapiteln beschrieben.

Die Anwender der POSYS 182x-B können zwischen diesen Funktionaltäten wählen, u.a. auch zwischen der Verwendung von Takt- & Richtungssignalen oder Signalen für die Ansteuerung von Mikroschrittmotoren. Diese Wahl beeinflußt auch die Wahl der Verstärker und der Anbindung an die Peripherie und eine Anzahl weiterer Funktionen. Die POSYS 185x-B sind ausgelegt nur Takt- & Richtungssignale auszugeben und daher sollte die Wahl der Treiber entsprechend ausfallen.

-o-

Bahngeneratorkontrolleinheiten

Für die POSYS® Schrittmotorprodukte gelten folgende Unterschiede, die POSYS® in Schrittmotormodus gibt Takt & Richtungssignale aus, die POSYS® in Mikroschrittmodus gibt Mikroschritte aus, dagegen sind in den Servoprodukten alle Einheiten Enkodereinheiten. Die nachfolgende Tabelle zeigt die Kommandos und die entsprechenden Einheiten.

Kommando	Servoachsen	Mikroschrittachsen	Takt- & Richtung Achsen
Set/GetPosition	counts	microsteps	steps
Set/GetVelocity	counts/cycle	microsteps/cycle	steps/cycle
Set/GetAcceleration	counts/cycle²	microsteps/cycle²	steps/cycle²
Set/GetDeceleration	counts/cycle²	microsteps/cycle²	steps/cycle²
Set/GetJerk	counts/cycle³	microsteps/cycle³	steps/cycle³
Set/GetStartVelocity	---	microsteps/cycle	steps/cycle
GetCommandedPosition	counts	microsteps	steps
GetCommandedVelocity	counts/cycle	microsteps/cycle	steps/cycle
GetCommandedAcceleration	counts/cycle²	microsteps/cycle²	steps/cycle²
Set/GetPositionErrorLimit	counts	microsteps	steps
GetPositionError	counts	microsteps	steps

Tabelle 60: Kontrolleinheiten

-o-

Enkoderrückmeldung

Die POSYS 185x-B oder 182x-B mit Achsen im Schrittmotormodus, unterstützen Enkoderrückmeldung in jeweils einem von zwei möglichen Wegen, entweder durch Verwendung von Inkrementalenkodern oder Absolutwertgebern mit SSI-Schnittstelle (optional EnDat oder BiSS). Das Kommando SetEncoderSource definiert die Art der Rückmeldung. Falls keine Rückmeldung verwendet wird (nicht ungewöhnlich bei der Verwendung von Schrittmotoren), dann sollte als Rückmeldung None mit dem Kommando SetEncoderSource gesetzt werden. Eine so konfigurierte Achse ignoriert die Rückmeldefunktion, dabei wird auch die Funktion Stall Detectiondeaktiviert, eine später noch detaillierter besprochene Funktion.

Egal welche Rückmeldemethode verwendet wird, die meisten Enkoderkommandos sind auch für Schrittmotoren gleich anzuwenden. z.B. die aktuelle Position kann mit dem Kommando GetActualPosition abgefragt werden, der Inhalt des Position Capture Registers kann mit GetCaptureValue gelesen und mit AdjustActualPosition und SetActualPosition kann die aktuelle Positionsangabe angepasst werden. Die Standardeinheiten für das Kommando sind Enkoderzählwerte. Mit dem Kommando SetActualPositionUnits kann das auf Schritte/Mikroschritte umgestellt werden. Die nachfolgende Tabelle zeigt die betroffenen Kommandos.

Command	Position Units = counts	Position Units = steps
Set/GetActualPosition	counts	steps/microsteps
AdjustActualPosition	counts	steps/microsteps
GetCaptureValue	counts	steps/microsteps

Tabelle 61: Enkoderrückmeldung

Das Kommando SetActualPositionUnits verändert auch die Einheiten der Tracevariablen Actual Position.

SetEncoderToStepRatio setzt das Verhältnis von Encodersignalen zur Anzahl Schritte pro Motorumdrehung um die Anzahl erfasste Enkodersignale in Schritte zu konvertieren. "Counts" ist die Enkoderauflösung pro Motorumdrehung. "Steps" ist die Anzahl Schritte pro Motorumdrehung. Da dieses Kommando ein Verhältnis definiert, müssen die Eingaben nicht eine volle Umdrehung repräsentieren um korrekte Werte zurückzuliefern, sofern das Verhältnis stimmt.

GetEncoderToStepRatio liest den programmierten Wert zurück.

-o-

Stall Detection

Zusätzlich zur passiven Positionserfassung durch den Host mit dem Kommando GetActualPosition, kann die POSYS® auch aktiv das Ziel und die aktuelle Position überwachen und einen Bewegungsfehler entdecken, der einen Stillstand zum Ergebnis hat. Die automatische Stillstandserfassung erlaubt den Controller festzustellen wann der Schrittmotor während der Bewegung Schritte verloren hat. Gewöhnlich passiert dies wenn der Motor auf ein Hindernis trifft oder wenn auf andere Art und Weise sein Nennmoment überschritten wird.

Die automatische Stillstandserfassung arbeitet nach der Initialisierung kontinuierlich. Die Sollposition wird mit der Istposition (Enkoder) verglichen und falls der Unterschied dieser beiden Werte einen vorher definierten Wert überschreitet wird ein sogenannter "Stall" erkannt. Im anwenderprogrammierten Register SetPositionErrorLimit wird der Schwellenwert bestimmt für den Bewegungsfehler (Stall).

Um die automatische Bewegungsfehlererkennung zu initiieren, muss vom Host aus die Anzahl Inkremente pro Schritt/Mikroschritt spezifiziert werden. Dies wird mit dem Kommando SetEncoderToStepRatio erfüllt. Dieses Kommando akzeptiert 2 Parameter. Der 1. Parameter ist die Anzahl Inkremente pro Motorumdrehung und der 2. Parameter ist die Anzahl Schritte/Mikroschritte pro Motorumdrehung.

Parameter	Format	Word size	Range
Encoder counts per rev	16.0	16-bit	0 to 32,767
Steps/microsteps per rev	16.0	16-bit	0 to 32,767

Tabelle 62: SetEncoderToStepRatio Parameter

Z.B. falls ein Schrittmotor mit 1,8° pro Vollschritt verwendet wird mit einem Enkoder mit 4000 Inkremente pro Umdrehung, müssten die Parameter wie folgt aussehen:

SetEncoderToStepRatio 4000 200 wobei die Anzahl Schritte pro Umdrehung sich wie folgt errechnet: 360/1,8.

Falls der gleiche Motor mit den Mikroschrittversionen der POSYS® verwendet wird und die Anzahl Mikroschritte pro Vollschritt ist auf 64 gesetzt, dann sehen die Parameter wie folgt aus:

SetEncoderToStepRatio 4000 12800 wobei die Anzahl Schritte pro Umdrehung wie folgt errechnet wird (360/1,8)*64.

In jenen Fällen in denen die Anzahl Schritte, Mikroschritte oder Inkremente pro Umdrehung das Maximum von 32767 überschreitet, können die Parameter als Bruchteile einer Umdrehung spezifiziert werden, solange das Verhältnis genau bewahrt wird. Von dem vorhergehenden Beispiel ausgehend kann das Verhältnis auch nachfolgend dargestellt werden.

SetEncoderToStepRatio 2000 6400

dabei das Verhältnis für eine halbe Umdrehung anzeigend. Das Verhältnis für einen Bruchteil einer Umdrehung anzugeben ist genau so präzise, wie man es für eine volle Umdrehung angibt.

-o-

Takt & Richtungssignalgenerierung

Falls bei der POSYS 182x-B eine Achse für Schrittmotorbetrieb eingestellt bzw. eine POSYS 185x-B verwendet wird, dann wird der Mechanismus für die Ausgabe von Takten aktiviert die auf den Treiber ausgegeben werden. Das Kommando für die POSYS 182x-B lautet hierfür SetMotorType. Bei der POSYS 185x-B ist nur diese Art der Signalausgabe möglich, das Kommando daher deaktiviert. Um für die POSYS 182x-B den eingestellten Modus zu lesen verwenden Sie das Kommando GetMotorType.

Das Taktsignal, dass vom Controller ausgegeben wird, besteht aus einer kontrollierten Serie individueller Takte die jedes für sich einen gewünschten Schritt repräsentieren. Dieses Signal wird immer als Rechtecksignal ausgegeben. Als Standard gilt ein Taktsignal als gegeben wenn das Signal von "low" auf "high" wechselt. Eine Invertierung dieser Signale wird später in diesem Kapitel beschrieben. Das Richtungssignal wird mit dem Taktsignal synchronisiert ausgegeben. Das Richtungssignal ist dahingehend kodiert, dass ein "high" für die positive und ein "low" für die negative Richtung gilt.

Die ausgegebene Taktrate wird bestimmt durch die entsprechenden Bahnprofilparameter. Jedoch unterstützt die

POSYS 1800-B unterschiedliche Bereiche der Taktgenerierung um die Ausgabe zu präzisieren für Geschwindigkeitsbereiche. Der zu verwendende Geschwindigkeitsbereich kann mit dem Kommando SetStepRange eingestellt werden. GetStepRange liest den eingestellten Wert.

Die folgende Tabelle listet die verschiedenen Möglichkeiten auf:

Kommando	Frequenzbereich für die auszugebenden Signale
SetStepRange 1	0 to 4.98 M steps per second.
SetStepRange 4	0 to 622.5 K steps per second.
SetStepRange 6	0 to 155.625 K steps per second.
SetStepRange 8	0 to 38.90625 K steps per second.

Tabelle 63: Wählbare Taktbereiche

Die Bereiche oben zeigen die minimalen und die maximalen Bereiche die auf dem Controller eingestellt werden können. Z.B. falls die gewünschte maximale Schrittrate 200K pro Sekunde ist, dann ist die korrekte Einstellung SetStepRange *4*.

Für Voll- und Halbschrittapplikationen, sowohl auch Takt- und Richtungsapplikationen die eine maximale Schrittrate von 38KSchritt/Sek. haben, sollte SetStepRange *8* verwendet werden. Für Applikationen die eine höhere Schrittrate als 38 KSchritt/Sek. erfordern, sollten höhere Einstellungen verwendet werden.

Für jede Achse kann eine individuelle Einstellung gewählt werden.

-o-

Mikroschritt Motor Kontrolle

Falls mit dem Kommando SetMotorType als Motortype Mikroschritt ausgewählt wurde, werden anstatt Takt- & Richtungssignale Signale für mehrere Phasen ausgegeben. Typische Schrittmotoren verfügen über 2 Phasen, manche haben auch drei Phasen. Die POSYS 182x-B unterstützt 2- und 3-Phasen Treiber. Welcher Wert eingestellt wurde, kann mit dem Kommando GetMotorType gelesen werden. Die POSYS 185x kann für diesen Modus nicht verwendet werden. Stattdessen verwenden Sie bitte Mikroschritttreiber die Takt- & Richtungssignale in Mikroschritte umwandeln.

Abbildung 37 zeigt einen Überblick über das Kontrollschema für Mikroschrittbetrieb.

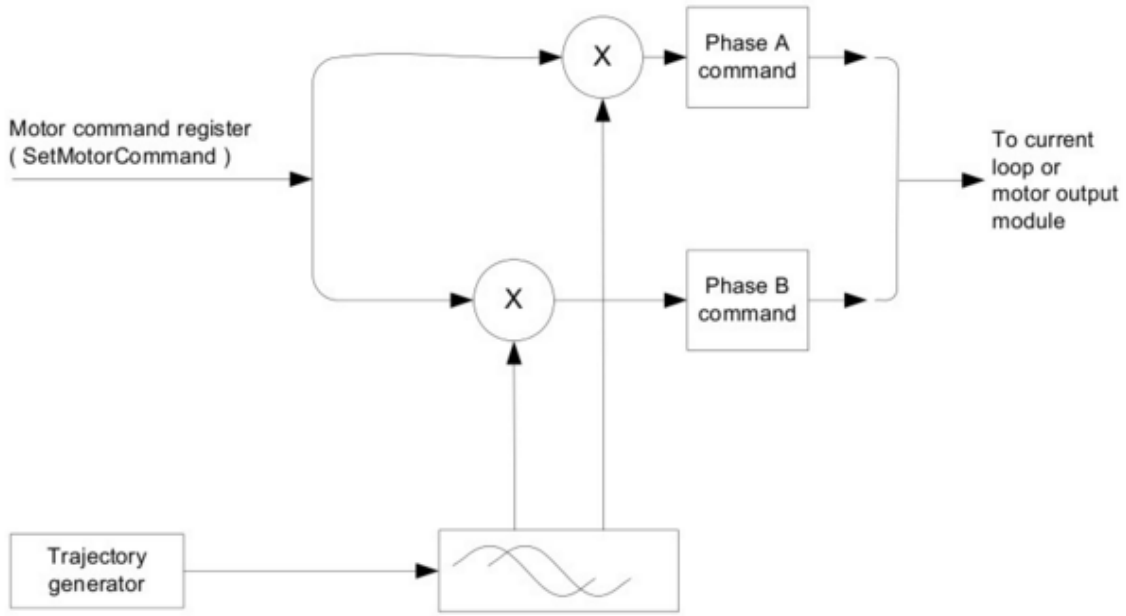

Abbildung 37: Mikroschrittsignalformerstellung

Der Mikroschrittteil des Reglers generiert ein sinusförmiges Signal mit einer Anzahl charakteristischer Ausgangswerte pro Vollschritt (ein Vollschritt ist ein Viertel einer elektrischen Periode). Die Anzahl von Mikroschritten pro Vollschritt

wird mit dem Kommando SetPhaseCounts definiert. Die Parameter für dieses Kommando repräsentieren die Anzahl Mikroschritte pro elektrischer Periode (4 Mal die erwünschte Anzahl von Mikroschritten). So sollen z. B. 64 Mikroschritte pro Vollschritt eingestellt werden, dann muss der Befehl SetPhaseCounts 256 benutzt werden. Die maximale Anzahl von Mikroschritten, die pro Vollschritt generiert werden können, ist 256. Der entsprechende Parameter lautet dann 1024.

Der Motorausgangspegel wird vom Motorbefehlsregister kontrolliert. Dieses Register kann mit dem Kommando SetMotorCommand gesetzt werden. Ein Wert zwischen 0 und 32767 repräsentiert eine Verstärkung zwischen 0% und 100%. Das Kommando SetMotorCommand ist doppelt gebuffert und benötigt daher zusätzlich ein Update -Kommando oder einen Breakpoint. Dieses Merkmal kann vorteilhaft sein, wenn es gewünscht wird, dass der Motorstrom sich ändert, dass man mit anderen Profiländerungen synchronisiert wie am Anfang oder dem Ende einer Bewegung.

Eine spezielle Begrenzung für das Halte-Kommando kann ebenso definiert werden um unterschiedliche Ausgabepegel für aktive und nicht-aktive Betriebsarten des Motors. Dies kann sinnvoll sein um die Hitzeentwicklung des Motors zu reduzieren wenn es sich nicht bewegt.

-o-

Mikroschrittsignalformen

Die POSYS 182x-B unterstützt zwei Mikroschrittmotortypen mit den ihnen eigenen Signalformen. Die eine Signalform ist die für 2-Phasen Schrittmotoren mit 90° Versatz, die andere unterstützt 3-Phasen Schrittmotoren mit 120° Versatz zwischen den Phasen. Um einen der beiden Motortypen zu spezifizieren verweden Sie das Kommando SetMotorType. Welche Einstellung vorliegt kann mit dem Kommando GetMotorType überprüft werden.

Zusätzlich sind für die verschiedenen Motortypen auch verschiedene Signalausgabemethoden verfügbar. Die folgende Tabelle listet diese auf.

Phasenanzahl	Motorausgabemodus	Anzahl Ausgangssignale und Name
2-Phasen Mikroschritt	PWMSign/Mag	2 (A, B)
2-Phasen Mikroschritt	DAC	2 (A, B)
3-Phasen Mikroschritt	PWM50/50	3 (A, B, C)
3-Phasen Mikroschritt	DAC	2 (A, B)

Tabelle 64: Mikroschrittausgangssignale

Die folgende Grafik zeigt die Phasen A und B für ein 2-Phasen Schrittmotor und die Phasen A, B und C Signale für ein 3-Phasen Schrittmotor.

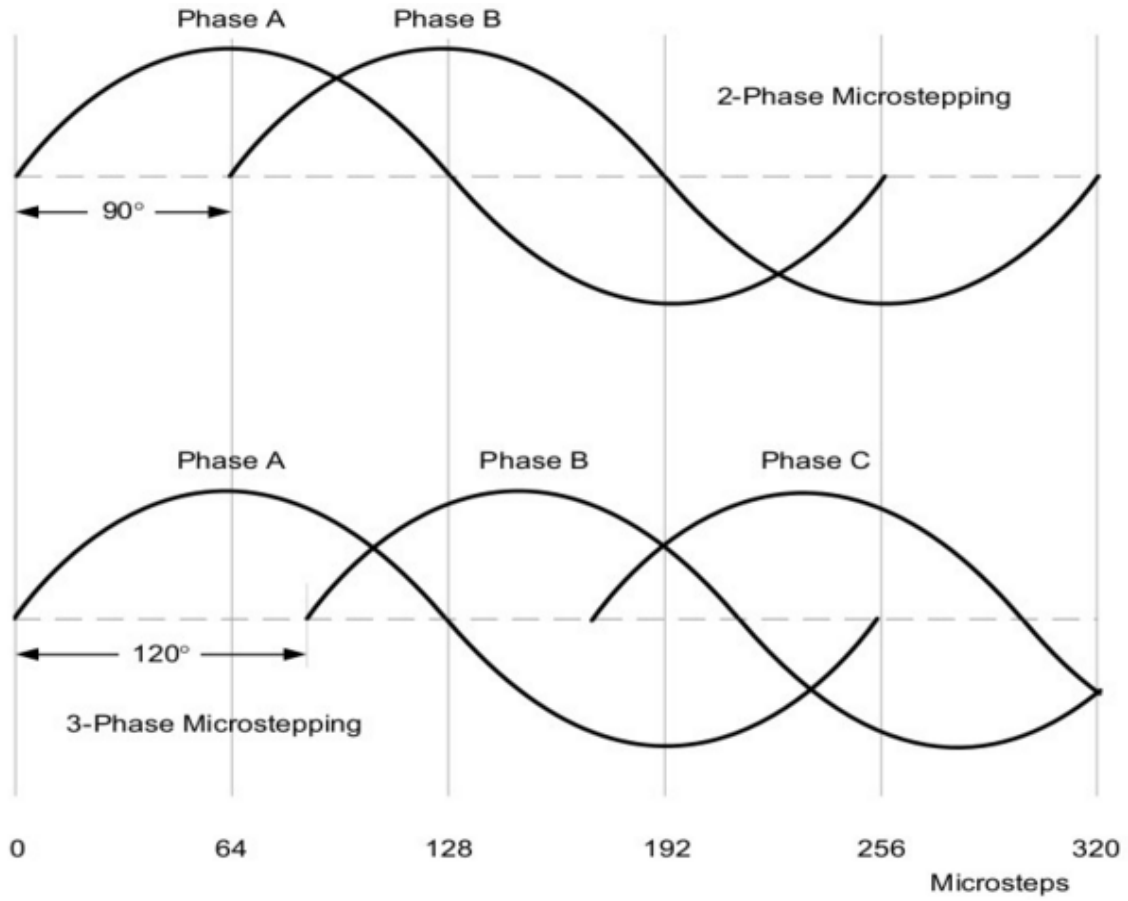

Abbildung 38: 2- und 3-Phasen Mikroschrittausgangssignale

-o-

Haltemoment Funktionen

Zusätzlich zu den Takt- & Richtungssignalen unterstützt die POSYS® für Schrittmotoren einen weiteren Ausgang für jede Achse, bekannt als das AtRest Signal, welches anzeigt ob ein Profil gerade abgearbeitet wird oder nicht. Dieses Signal kann hilfreich sein in Verbindung mit Treibern die ein anderes Moment im Stillstand aufweisen als während der Bewegung.

Für Achsen die im Mikroschrittmotormodus verwendet werden, existiert eine ähnliche Funktion die es erlaubt einen spezifischen Haltemoment zu definieren. Der Strom der während des Mikroschrittbetriebes benutzt wird, wird definiert mit dem Kommando SetMotorCommand. Falls gewünscht ist es möglich im Haltezustand die Signalausgabe zu begrenzen. Dieser Grenzwert wird mit dem Kommando SetHoldingCurrent gesetzt. Welcher Wert gesetzt ist, kann mit dem Kommando GetHoldingCurrent gelesen werden. Bitte beachten Sie dass der Wert den Ausgabestrom begrenzt sobald der Motor sich in der Ruhephase befindet. z.B. falls der Ausgabewert für den Treiber der durch das Kommando SetMotorCommand definiert wurde, bereits niedriger ist als der Wert der mit dem Kommando SetHoldingCurrent, wird der niedrigere Wert verwendet.

Egal ob durch das AtRest-Signal oder durch das Kommando SetHoldingCurrent definiert, die Haltemomentbedingung wird aktiviert sobald der Bahngenerator einen Geschwindigkeitswert von Null (0) aufweist für einen vom Anwender programmierten Zeitrahmen. Falls dieser Parameter einen anderen Wert als Null (0) aufweist, erlaubt es eine Verzögerung einzustellen zwischen dem Zeitpunkt zu dem der Bahngenerator seine Aufgabe erledigt hat und das externe Signal aktiviert wird. Dies wird in der Regel eine Einstellung aufweisen, die es dem Motor erlaubt zu einem kompletten Stillstand zu kommen. Mit dem Kommando SetHoldingCurrent kann diese Verzögerung eingestellt werden. Der eingestellte Wert kann mit dem Kommando GetHoldingCurrent zurückgelesen werden.

-o-

On-Board Speicher: Definition und Zugriff

Speicherkonfiguration

Die POSYS® 1800-B ist in der Lage On-Board Speicher zu adressieren um Tracedaten zu speichern. Zusätzlich kann es verwendet werden um andere Daten zu hinterlegen, wie z.B. Profilsequenzen (SetProfileMode ExternalProfileMode).

Der MC ist in der Lage, große Mengen an Bewegungsdaten in externem RAM zu verwalten. Aufgrund schnelleren Datenaustauschs wird in der Regel ein Dual Port Memory eingesetzt, welches von beiden Seiten MC und PC konkurrierend bedient wird. Auf der Baugruppe POSYS 1800 B wurde dieses DPM innerhalb des FPGAs realisiert. Die Größe beträgt 32768 Words zu 16 Bit.

Innerhalb dieses Speicherbereichs können vom MC Datenpuffer angelegt werden, die vom PC zu geeigneten Zeitpunkten geleert oder gefüllt werden müssen. Zur Erleichterung dieser Aufgabe wurden für den Lese- und den Schreibzugriff je ein Adressregister eingeführt, welches eine PC-seitige Interruptanforderung auslösen kann, sobald ein entsprechender MC-seitiger Zugriff erfolgt. Die Schreib- und Leseadressen müssen durch die Datentransferroutinen im PC verwaltet werden.

Da für DPM-Schreib- und Lesehits, sowie für MC Host Interrupts nur ein PC-Interrupt verwendet wird, muß die Identifizierung der Interruptquelle innerhalb der POSYS 1800-B möglich sein. Dazu steht das Interrupt Statusregister 0x0234 im FPGA Subsystem zur Verfügung.

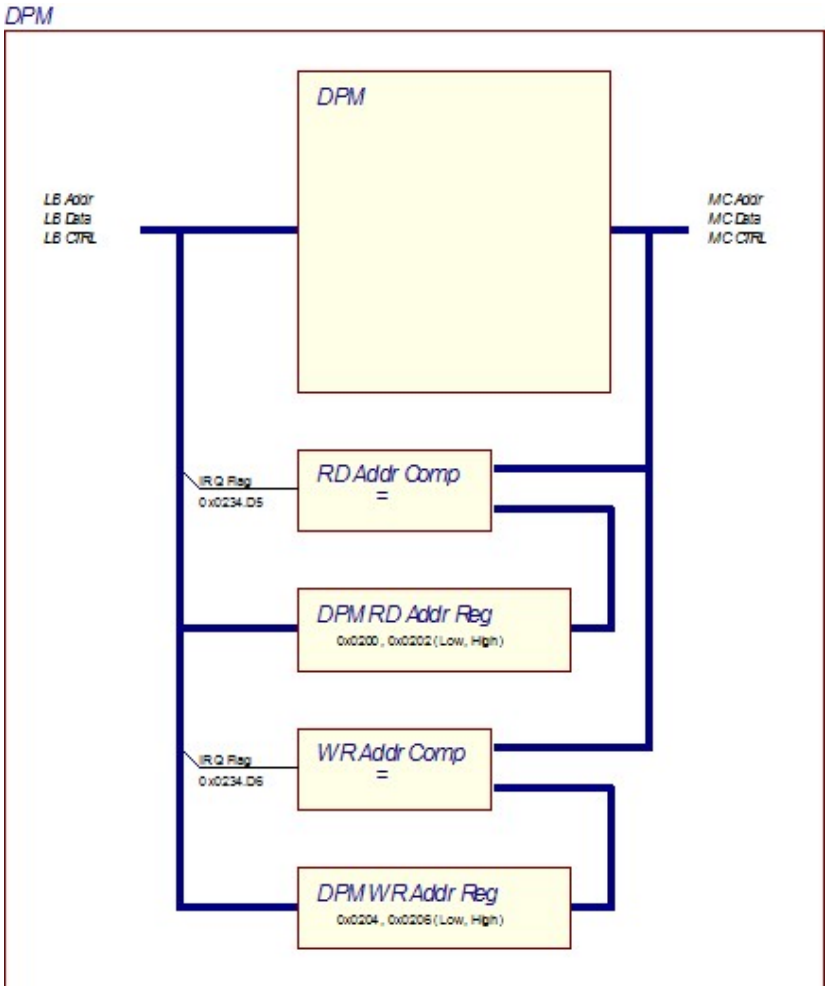

Abbildung 39: DPRAM

-o-

PC104 ISA Bus (IO)

Hier sind MC Systemsteuerregister, MC Host Port, User Register, DPM Threshold Register und SPI DAC Register gemappt. Der Zugriff durch den PC erfolgt, wie von früheren POSYS Versionen bekannt, über die 16 Bit Port IO Befehle INPORT und OUTPORT. Der PC läßt eine Byte- Adressierung zu, es sind jedoch nur Word-Zugriffe auf geradzahlige Adressen im bereich 0x0000…0xFFFE erlaubt.
Im IO-Bereich ist außerdem ein Index- und Datenregister gemapped, mit dem der Zugriff auf die FPGA-internen Subsysteme DPM, DAC I/F und User LED möglich ist.

-o-

PC104 ISA Bus (Speicher)

Hier ist der PC-seitige DPM-Port als externer Arbeitsspeicher gemappt. Der Zugriff erfolgt über 16 Bit Memory Read-/Write-Befehle. Die Basisadresse ist über SW200 zwischen 0x8'0000 und 0xF'0000 in Schritten von 0x1'0000 (64kByte) einstellbar. Die lineare Adressdarstellung 0x8'0000 entspricht dabei der Segment:Offset- Adresse 0x8000:0000. Der 64kByte DPM Adressraum kann mit konstantem Segment und variablem Offset vollständig adressiert werden. Der PC läßt eine Byte-Adressierung zu, es sind jedoch nur Word-Zugriffe auf geradzahlige Adressen erlaubt.
Die Deaktivierung des Memory Adressraums erfolgt durch setzen einer Mem Base Addr von < 0x8'0000 (SW200.8 = 0). In diesem Fall kann der DPM Zugriff immer noch über das IO-gemappte FPGA Subsystem erfolgen.

-o-

MC Bus

Hier ist der MC-seitige DPM-Port als externer Speicher gemappt. Außerdem befinden sich hier die DAC Output Register und der MC User Space, der über den MC Host Port mit den Subadressen 0x00 … 0xFF angesprochen werden kann. Der MC unterstützt keine Byte-Adressierung. Alle Adressen sind Word-Adressen und können auch ungeradzahlig sein. Die Umrechnung zwischen MC-seitigen und PC-seitigen DPM-Adressen erfolgt somit über den Faktor 2. Der MC-seitige DPM Adressraum erstreckt sich von 0x0000 … 0x7FFF.

-o-

Externe Speicherpuffer

Die POSYS 1800-B Motion Controller bietet eine Reihe von Komm andos, die verwendet werden, um auf den ex ternen Speicher zu zugreifen. Dieser Raum ist in einzelne Puffer eingeteilt, um einen besseren flexibleren Zugang zu bieten. Es erlaubt bis zu 32 Speicherpuffer zu definieren. In diesem Multi-Puffer Schem a stellt ein Puffer einen zusamm enhängenden Block im Speicher dar und wird durch die Definition einer Basis-Adresse für den Block und einer Blocklänge beschrieben. Sobald ein Pufferbasisadresse und -Länge definiert w urden, können Datenw erte in und aus dem Puffer geschrieben und gelesen w erden.

Bei der Definition der Speicherpuffer w ird der externe Speicher als eine Folge von 32-Bit-Speicherplätzen behandelt. Jede 32-Bit-Wert nimm t bis zu zwei 16-Bit-Speicherplätze im physikalischen Speicher ein. Beides, die Pufferbasisadressen und -Längen arbeiten m it 32-Bit-Werten und m üssen deshalb verdoppelt w erden um die entsprechenden physikalischen Adressen zu erhalten. Bei der Definition von Speicherpuffer, erlaubt der Motion-Controller jedw ede Werte für die Basisadresse und -Länge, solange die Werte physikalisch korrekt sind und existieren. Korrekte Speicheradressen reichen von 0 bis 3FFF FFFFh (entsprechende physikalische Adresse 7FFF FFFEH). Sofern die vollen zw ei gigawords physikalischer Speicher vorhanden sind, ist es m öglich den Speicherpuffer der Karte in einen Speicherbereich zu mappen der k einen physik alischen Speicher enthält.

Zusätzlich zur Basis-Adresse und -Länge, verw altet jeder Speicherblock einen Lese-Index und einen Schreib-Index. Der Lese-Index kann einen Wert zw ischen 0 und L-1 zugeordnet w erden (wobei L die Länge des Puffers ist). Es definiert den Ort, von dem der nächste Wert abgelesen w erden kann. Ebenso schreibt der Index von 0 bis L-1 und definiert den Ort, an dem der nächste Wert geschrieben w erden k ann. Wenn ein Wert aus dem Speicherpuffer gelesen w ird, wird der Lese-Index automatisch erhöht, dabei den nächsten Wert für das Lesen definierend. Der Schreib-Index w ird jedesm al erhöht, wenn ein Wert in den Puffer geschrieben w ird. Sobald eines der Index e das Ende des Puffers erreicht, w ird es autom atisch auf 0 zurückgesetzt für die nächste Lese-/Schreib-Operation.

-o-

On-Board Speicherkommandos

Dieser Abschnitt beschreibt die Kommandos um auf den Speicherbereich zuzugreifen und Schreib- und Lesevorgänge zu vollziehen.

SetBufferStart bufferID, address
Setzt die Basisadresse des Zwischenspeichers. bufferID ist ein 16-Bit Integerwert im Bereich von 0-15, welches an gibt, welchen Zwischenspeicher es modifizieren soll, address ist ein 32-Bit Integerwert, welches die neue Basisadresse des Zwischenspeichers definiert. Der Controller addiert address zur gegenwärtigen Zwischenspeicherlänge (wie in SetBufferLength definiert) um sicher zu stellen, dass der Zwischenspeicher den maximal adressierbaren Speicherbereich nicht überschreitet. Falls die Grenze überschritten wird, wird das Kommando ignoriert und ein Instruction Error Bit gesetzt.

GetBufferStart bufferID
Liefert die Basisadresse des abgefragten Zwischenspeichers

SetBufferLength, bufferID, length
Definiert die Länge des Zwischenspeichers. bufferID ist ein Integerwert im Bereich von 0-31. length ist ein 32-Bit Integer im Bereich von 1 bis 3FFFFFFFh. Der C ontroller addiert length zur gegenwärtigen Zweischenspeicherbasisadresse (wie in SetBufferStart gesetzt) um sicher zu stellen, dass der maximal zur Verfügung stehende Speicherbereich nicht überschritten wird. Falls doch, wird das Kommando ignoriert und das Instruction Error Bit wird gesetzt.

SetBufferStart und SetBufferLength setzen die Bufferindexe wieder auf Null (0) zurück.

GetBufferLength bufferID
Liefert die Länge des spezifizierten Buffers.

SetBufferReadIndex bufferID, index
Setzt den Schreibindex für den spezifizierten (bufferID) Buffer. Index ist ein 32-Bit Integerwert von 0 bis length-1, wobei length (mit SetBufferlength gesetzt) die momentane Bufferlänge definiert. Falls index nicht innerhalb dieses Bereiches liegt wird es nicht gesetzt und ein Instruction Error Bit wird gesetzt.

GetBufferReadIndex bufferID
Liefert den Wert des Schreibindexes (bufferID) für den angegebenen Zwischenspeicher.

SetBufferWriteIndex bufferID, index
Setzt den Leseindex für den spezifizierten Buffer (bufferID). index ist ein 32-Bit Integerwert von 0 bis length-1, wobei length die momentane Bufferlänge definiert. Falls index nicht innerhalb dieses Bereiches liegt wird es nicht gesetzt und ein Instruction Error Bit wird gesetzt.

GetBufferWriteIndex bufferID
Liefert den Wert des Schreibindexes für den angegebenen Zwischenspeicher.

ReadBuffer bufferID
Liefert einen 32-Bit Wert vom spezifizierten Zwischenspeicher zurück. Die Position von dem dieser Wert gelesen wird, wird bestimmt indem zum Leseindex die Basisadresse hinzuaddiert wird. Nachdem der Wert gelesen wurde, wird der Leseindex inkrementiert. Falls das Ergebnis gleich der gegenwärtigen Bufferlänge entspricht wird der Leseindex auf Null zurückgesetzt.

WriteBuffer bufferID, value
Schreibt einen 32-Bit Wert in den spezifizierten Zwischenspeicher. Die Position wird bestimmt indem zum Schreibindex die Basisadresse hinzuaddiert wird. Nach dem der Wert geschrieben wurde, wird der Schreibindex inkrementiert. Falls das Ergebnis gleich der gegenwärtigen Bufferlänge entspricht, wird der Schreibindex auf Null zurückgesetzt.

-o-

User I/O

Zugriff auf Karten-spezifische Funktionalitäten erhält man durch das für allgemeine Zwecke gedachte I/O-Adressierungsschema. Die Kommandos ReadIO und WriteIO erlauben den Zugriff auf diese spezifischen Funktionen, die von der POSYS® unterstützt werden. Für weitere Informationen bzgl. der ReadIO/WriteIO Kommandos verweisen wir auf die Zusatzbeschreibung in der Sektion FAQs/Programmer's Info und auf die Beschreibung der beiden Kommandos im Programmierhandbuch (ReadIO/WriteIO).

-o-

Kommandoübersicht - ReadIO/WriteIO

Dieses Kapitel beschreibt die Details um die Host I/O Kommandos die die POSYS 1800-B veranlassen auf die periphären Geräteadressen zu zugreifen.

ReadIO address
liest ein 16-Bit Datenwort vom Gerät dessen Adresse kalkuliert wird indem zur Adresse 1000H addiert wird. (address ist ein Offset der base address, 1000h, der POSYS® memory-mapped I/O space).

WriteIO address, data
schreibt ein 16-Bit Datenwort in ein Gerät dessen Adresse sich daraus errechnet indem 1000h der Adresse hin zu addiert wird. (address ist ein Offest der Basisadresse, 1000h, vom memory-mapped I/O Bereich der POSYS®).

Die folgende Tabelle listet die verschiedenen Adressierungen und dadurch erhältlichen Funktionen auf:

I/O space address offset	Register	Beschreibung
+0	Digital I/O	Dieses Register erlaubt das Schreiben und Lesen dieser 8 für allgemeine Zwecke gedachten digitalen Ein- und Ausgänge.
+1	Amplifier & DAC enable	Dieses Register erlaubt das Schreiben und verifizieren der Ausgangssignale für die Verstärkerfreigaben 1-4. Diese Ausgänge können auch für allgemeine Zwecke verwendet werden. Ebenso werden mit diesem Register die DACEnable Ausgänge beschrieben und verifiziert.
+2	Reset monitor	Register um die Überwachung zu checken und zu resetten
+3	unbenutzt	-
+4	Watchdog	Register um die Watchdog Funktion zu aktivieren und upzudaten.
+0xff	Card ID	Register um das KartenID-Wort zu lesen.

Tabelle 65: I/O Space Address Offset

-o-

Digital I/O-Kontrollregister (I/O space address +0)

Die folgende Tabelle beschreibt Details für das +0 Digital IO Kontrollregister.

I/O Address	Bitwert	Signale
+0	0-7	DigitalOut0-7
	8-15	DigitalIn0-7

Tabelle 66: Digitales E/A Kontrollregister

-o-

Verstärker- & DAC Freigabekontrollregister (I/O space address +1)

Die folgende Tabelle beschreibt die Details für das +1 Verstärker- & DAC Freigabe Kontrollregister.

I/O Address	Bitwert	Signale
+1	0-3	Verstärkerfreigabeausgänge (0-3)
	4-6	nicht benutzt
	7	DAC Freigabestatus (1 = aktiviert, 0 = deaktiviert)
	8-11	Maske ändern für Bits 0-3 - Verstärkerfreigabeausgänge (1 = ändern, 0 = nicht ändern)
	12-14	nicht benutzt
	15	Maske ändern für DAC Freigabe (1 = ändern, 0 = nicht ändern)

Tabelle 67: Verstärker & DAC Freigaberegister

-o-

Reset Überwachungskontrollregister (I/O space address +2)

Die folgende Tabelle beschreibt die Details für das +2 address Reset Überwachungskontrollregister.

I/O Address	Bitwert	Signale
+2	0-11	reserviert
	12	Softwarekommando: eine 1 in diesem Bit zeigt an, dass ein Reset durch ein Anwendersoftwarekommando ausgelöst wurde.
	13	Under voltage detection: eine 1 in diesem Bit zeigt an, dass ein Reset aufgrund einer Unterspannungserkennung ausgelöst wurde.
	14	External signal: eine 1 in diesem Bit zeigt an, dass ein Reset aufgrund eines externen Signals ausgeführt wurde. Ein solches Reset kann z.B. über das GP Con herrühren, dass mit dem externen Resetknopf des IO700/800 verbunden ist. Wenn dieses Signal auf „low“ geht, wird ein Reset ausgelöst.
	15	Watchdog timeout: eine 1 in diesem Bit zeigt an, dass ein Reset aufgrund eines Watchdog Timeouts erfolgte.

Tabelle 68: Reset Überwachungskontrollregister

-o-

Card ID-Kontrollregister (I/O space address +0xff)

Die folgende Tabelle beschreibt die Details für das +0xff address card ID Register.

I/O Address	Bitwert	Signale
+0xff	0-3	Major PLD Revision: binär-kodiertes 4-Bit Wort, welches die Major PLD Revision angibt. Dieser Wert reicht von 0-15.
	4-7	Minor PLD Revision: binär-kodiertes 4-Bit Wort, welches die Minor PLD Revision angibt. Dieser Wert reicht von 0-15.
	8-11	Board revision: binär-kodiertes 4-Bit Wort, welches die Board Revision angibt. Dieser Wert reicht von 0-15.
	12-15	Boardtyp: binär-kodiertes 4 Bit Wort, welches die Boardtype angibt. Dieser Wert kann eine der folgenden Werte haben: 0 – ISA-Bus basierte Kartenfamilie 1 – PCI-Bus basierte Kartenfamilie 2 – unbenutzt 3 – PC/104 basierte Kartenfamilie 4 – 15 unbenutzt

Tabelle 69: Card ID Kontrollregister

-o-

Watchdog Kontrollregister (IO space address +4)

Die folgende Tabelle beschreibt die Details für das +4 address Watchdog Timeout Register. Ein Scheibkommando mit WriteIO aktiviert den Watchdog Timer. Anschliessend muss alle 104 millisec. in dieses Register geschrieben werden um ein Reset des Controllers zu vermeiden. Einmal aktiviert kann es nicht mehr daktiviert werden, ausser durch ein PowerOff/PowerOn Vorgang.

I/O Address	Bitwert	Signal
+4	0-15	0x5562

Tabelle 70: Watchdog Kontrollregister

-o-

Index

- 1 -

16 Bit DAC Signalausgabe 73

- 5 -

50/50 PWM 72

- A -

Absolutenkodereingang (SSI, BiSS, EnDat) 69
Activity Status Register 46
Address Bit Protocol 85
Aktivieren des Motor Output Modules 70
Aktivieren und Deaktivieren von Control Modules 10
Aktivieren und Deaktivieren von Trajectory Generator Modules 25
Aktuelle Positionsregister 66
Algorithmische Initialisierungssequenz 94
Algorithmische Phaseninitialisierung 92
Analoge Eingänge 64

- B -

Bahnen, Profile und Parameter 14
Bahngeneratorkontrolleinheiten 99
Bestimmung des Bi-Quad Skalierungsfaktors 32
Biquad Output Filter 31
Biquadkoeffizienten bestimmen 32
Blockschaltbild 9
Breakpoint Aktionen 41
Breakpoint Beispiele 42
Breakpoint Latenzen 42
Breakpoint Trigger 40
Breakpoints 39

- C -

CAN Ereignis Benachrichtigung 88
Card ID-Kontrollregister (I/O space address +0xff) 109
Checksummen 83
Control Flow Überblick 10
Controller Area Network (CAN) 86

- D -

Das AxisOut-Signal 64
Das SetStop Kommando 25
Das Zeitregister 12
Datenerfassung Starten 61
Deaktivieren und Aktivieren des Position Loop Modules 36
Definition der Bahnparameter 14
Der AxisIn-Eingang 64
Der Trace Buffer 57
Der Trace Period 57
Derivative Sampling Time 28
Digital I/O-Kontrollregister (I/O space address +0) 108
Digitale Filterung 67
Direct-Set Initialisierungssequenz 94
Direct-Set Phaseninitialisierung 93
Download von Trace Daten 60
Dual Encoder PID Loop Algorithmus 29
Dual Encoder Support 29

- E -

Einen Breakpoint definieren 39
Einstellung der PWM-Frequenz 73
Electronic Gear Profil 20
Encoder Prescaler 96
Endschalter 51
Enkoderrückmeldung 100
Enkoderschnittstelle 66
Event Status Register 45
External Profile Mode 21
Externe Speicherpuffer 106

- G -

Geschwindigkeitsmodus 19
GetVersion Kommando 13
Gewährleistung 7

- H -

Haftungsausschluß 7
Hallsensor-basierte Initializierungssequenz 93
Hallsensor-basierte Phaseninitialisierung 91
Haltemoment Funktionen 104
Hardware Control Signals 64
High-Speed Position Capture 68
Host I/O Commands 80
Host Interrupts 61
Hostkommunikation 80

- I -

Idle-line Mode 85
Impuls-Breiten-Modulierte (PWM) Motorsignalausgabe 77
In-Motion Indikator 55
Index-Taktreferenzierung 94
Inkrementaler Enkodereingang 66
Instruktionsfehler 46, 81, 83
Integration Limit 28

- J -

Justierung der Phasenwinkel 95

- K -

Kommandointerpretation 82
Kommandoübersicht - ReadIO/WriteIO 108
Konfiguration 82
Konfiguration der CAN Schnittstelle 87
Konfiguration von Dual-Enkoder Unterstützung 30

- L -

Level-Triggered Breakpoints 41

- M -

MC Bus 106
Message Format 86
Mikroschritt Motor Kontrolle 102
Mikroschrittmotorsignalausgabe 77
Mikroschrittsignalformen 103
Modellversionen Überblick
Motion Complete Indikator 54
Motor Bias 35
Motor Output Module deaktivieren 70
Motorausgabeformat 71
Motorausgabesignale 70
Motorausgangssignal Interpretation 77
Motortreiber-Konfigurationen 78
Motortyp 70
Multi-drop/Idle-line Mode 84
Multi-Phasen Motoranbindung 74
Multi-Phasen Motorkommandointerpretation 75
Multi-Turn Systeme 69

- O -

On-Board Speicherkommandos 107

Output Limit 34
Output Scaling 28

- P -

Paralleler Kommunikationsport (ISA-Bus) 80
Parameter Buffering 38
PC104 ISA Bus (IO) 106
PC104 ISA Bus (Speicher) 106
Phasenanzahl 89
Phasenfehlererkennung 95
Phaseninitialisierung 91
Phasenkontrollmodule 90
Phasenzählungen 90
PID Loop 26
Point-to-Point Mode 84
Position Loop Werte lesen 37
Profile Mode 22
Programmierung der Phaseninitialisierung 93

- R -

Relevante On-Board RAM-Funktionen 22
Reset Kommando 11
Reset Überwachungskontrollregister (I/O space address +2) 109

- S -

S-Kurven Punkt-zu-Punkt Profil 17
Schleppabstandsfehler 51
SetEventAction Prozess 50
Settle Window 55
Sicherheitshinweis 7
Sign/Magnitude PWM 72
Signal Sense Mask 48
Signal Status Register 47
Sinuskommutierung 98
Skalierung der Bi-Quad Koeffizienten 33
Speicherkonfiguration 105
Stall Detection 101

- T -

Takt & Richtungssignalgenerierung 101
Takt- & Richtungssignalausgabe 76
The Synch Pin - Multiple Card Synchronization 65
Threshold-Triggered Beakpoints 41
Trace Buffer Architektur 56
Trace Capture 56
Trace Mode 23
Trace Modes 59
Trace Start/Stop Bedingungen 59
Trace Variablen 58

Tracking Window 53
Trapezfömiger Punkt-zu-Punkt Profil 15

- U -

Updates 38
User I/O 108

- V -

Verfügbarer Speicher (RAM) zu klein für Programmausführung! Was tun? 24
Verstärker- & DAC Freigabekontrollregister (I/O space address +1) 108
Verwendete Symbole 8
Verwendung von On-Board RAM für External Profile Mode 23

- W -

Was bedacht werden muß 24
Watchdog Kontrollregister (IO space address +4) 110

- Z -

Zykluszeit setzen 12

- Ä -

Änderungen zu vorhergehenden Handbüchern 8

- Ü -

Überblick 26, 45, 86, 89, 99
Übertragungsprotokolle 84

© POSYS Motion Control, 2014
www.halbeck.com